Prust – Voß

APPLE II
Tips & Tricks

Eine Fundgrube für den APPLE II Anwender

EIN DATA BECKER BUCH

Prust – Voß

APPLE II
Tips & Tricks

Eine Fundgrube für den APPLE II Anwender

EIN DATA BECKER BUCH

ISBN 3-89011-025-8

Copyright (C) 1984 DATA BECKER GmbH
Merowingerstr. 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Wichtiger Hinweis!

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technische Angaben und Programme in diesem Buch wurden von den Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler sind die Autoren jederzeit dankbar.

Vorwort

Mit seinen über 400 Seiten halten Sie das derzeit umfangreichste DATA BECKER BUCH in Ihren Händen. Gleichzeitig ist es unser erstes zum APPLE II (beziehungsweise IIc, IIe).

Werner Voß und Renate Prust geben mit diesem Buch dem APPLE-Anwender viele Hinweise und Hilfen zum Selberprogrammieren. Ob es sich nun um Peripherie, dreidimensionale Grafik oder wichtige Speicheradressen handelt, etwas nützliches ist mit Sicherheit für jeden dabei. Gleichzeitig ist dieses Buch nicht nur eine Fundgrube für Tips & Tricks, sondern durch seine klare Gliederung gleichzeitig ein Arbeits- und Handbuch zur Programmierung in APPLESOFT-BASIC.

Viel Erfolg beim Programmieren!

Dr. Achim Becker

INHALTSVERZEICHNIS Seite

Vorbemerkungen 1

Kapitel 1: Grundlegende BASIC-Elemente 7

1.1 Einführung 7
1.2 BASIC 8
1.3 Einfache Ausgabe- und Eingabeanweisungen 1o
1.4 Erste Kommandos 15
1.5 Programmverzweigungen 16
1.6 Programmschleifen 19
1.7 Dateneingabe 2o
1.8 Stringbearbeitungsfunktionen 23
1.9 Benutzung von Disketten 25
1.1o Ergänzungen 27
1.11 Abschlußbeispiel 29

Kapitel 2: Die Nutzung peripherer Geräte 41

2.1 Vorbemerkungen 41
2.2 Der Drucker 41
2.2.1 Ausgabe von Programmen 43
2.2.2 Ausgabe von Programmergebnissen ... 44
2.2.3 Graphiken 45
2.3 Diskettenlaufwerk 51
2.3.1 Vorbemerkungen 51
2.3.2 Tips bei der Initialisierung 55
2.3.3 Interaktive Diskettennutzung 56
2.3.4 Beispiel: Statistische Auswertung . 67

Seite

Kapitel 3:	Speicherzugriff und Maschinenprogramme	8o
3.1	Direkter Speicherzugriff	8o
3.2	Die Benutzung von ASSEMBLER-Programmen	111
3.2.1	Grundlagen	111
3.2.2	Verknüpfungen BASIC-ASSEMBLER	129
3.3	Unterprogramme in Maschinensprache .	14o
3.4	Anwendungsbeispiele	146
Kapitel 4:	Graphikprogramme	156
4.1	Vorbemerkungen	156
4.2	Normalgraphik	158
4.2.1	Textmodus und Graphikmodus	158
4.2.2	Farben und graphische Effekte	161
4.2.3	Graphik-Anweisungen	163
4.2.4	Beispiel 1: Bildschirmrand	165
4.2.5	Beispiel 2: Gitter	166
4.2.6	Beispiel 3: Farbdemonstration	167
4.2.7	Beispiel 4: Histogramm	168
4.2.8	Beispiel 5: Ballspiel	171
4.3	Hochauflösende Graphik	174
4.3.1	"Einschalten" der hochauflösenden Graphik	176
4.3.2	Farben	177
4.4.3	Graphik-Anweisungen	178
4.3.4	Beispiel 1: Koordinatensystem	179
4.3.5	Beispiel 2: Lineare Funktion	18o
4.3.6	Beispiel 3: Kreis	183
4.3.7	Beispiel 4: Dreidimensionale Graphik	186

Seite

4.4.	Das Mischen von Text und Graphik ...	188
4.4.1	"Zeichnen" von Symbolen	189
4.4.2	Hilfsprogramme zum Mischen von Text und Graphik	193
Kapitel 5:	Verbesserung der Programmstrukturen	195
5.1	Vorbemerkungen	195
5.2	Unterprogrammtechnik	197
5.3	Menütechnik	2o9
5.4	Mischen von Programmteilen	22o
Kapitel 6:	Editieren	238
6.1	Vorbemerkungen	238
6.2	Der Escape-Modus	239
6.3	PROGRAM LINE EDITOR	245
Kapitel 7:	Programmierhilfen	263
7.1	Vorbemerkungen	263
7.2	Umnumerieren und Mischen von Programmen	265
7.3	Überblick und Dokumentation	287
7.3.1	Variablenkontrolle	289
7.3.2	Kontrolle der Sprungadressen	295
7.3.3	Änderungen	298
7.4	Initialisierung	3o3
7.5	Bildschirmgestaltung	322
7.5.1	Positionierung auf dem Bildschirm ..	322

Seite

7.5.2 Stringfunktionen 325
7.5.3 Formatierte Bildschirmausgabe 328
7.5.4 Bildschirmmasken 334
7.6 Sortieren 342

Kapitel 8: Alternativen zur BASIC-Programmierung 351

8.1 Vorbemerkungen 351
8.2 Andere Programmiersprachen 353
8.3 Software 38o

Anhang I: Fehlermeldungen (Auswahl) 386

Anhang II: Wichtige Unterprogrammadressen 39o

Anhang III: Speicheradressen (Auswahl) 391

Anhang IV: Speichereinteilung des APPLE II ... 393

Stichwortverzeichnis 394

Vorbemerkungen

Jeder, der sich mit einem Kleinrechner, wie z.B. dem APPLE II, intensiver beschäftigt, wird über kurz oder lang feststellen, daß eine Reihe von typischen Anwenderfragen in den mitgelieferten Handbüchern oder in der einführenden Mikrocomputerliteratur zu kurz kommen.

Es ist deshalb Anliegen dieses Buches, einige Tips und Tricks vorzustellen, die die Programmierung des APPLE II erleichtern oder auf geschickte Weise einige Probleme lösen.

Es versteht sich von selbst, daß bei der Auswahl derjenigen Anwenderprobleme, die hier besprochen werden, ein gewisses Maß an Willkür nicht zu vermeiden ist. Es gibt ja unendlich viele spezielle Probleme, aber der zur Verfügung stehende Platz zwingt dazu, eine Auswahl vorzunehmen. Wir haben diese Auswahl so vorgenommen, wie sie sich aufgrund unserer mehrjährigen Erfahrungen beim Einsatz des APPLE II bei konkreten Problemstellungen angeboten hat:

Es zeigt sich nämlich beim praktischen Rechnereinsatz sehr schnell, daß sich bestimmte Problemtypen recht häufig wiederholen - und genau diese haben wir uns vorgenommen.

Bevor der Leser sich der Erörterung dieser Probleme nun widmen kann, sind allerdings einige Vorbemerkungen notwendig, wenn die später vorgestellten Programme nutzbringend verwendet werden sollen.

1. In den Vordergrund der Betrachtungen werden Programme in der Programmiersprache BASIC gestellt. Auf andere Sprachen wird nur in wenigen Kapiteln Bezug genommen.

2. Wir gehen davon aus, daß der Leser schon über einige BASIC-Programmiererfahrung verfügt.
   Um aber auch dem Anfänger die Möglichkeit zu bieten, die hier vorgestellten Programme nachzuvollziehen oder verwenden zu können, stellen wir den eigentlichen Ausführungen ein Kapitel voran, welches die grundlegenden BASIC-Anweisungen wiederholt.

   Der "Könner" auf diesem Gebiet mag dieses Kapitel getrost überblättern.

3. Bei den einzelnen Problemstellungen, die hier aufgegriffen werden, verwenden wir nur in Ausnahmefällen Software-Angebote oder speziell für den APPLE II zur Verfügung stehende Dienst- oder Hilfsprogramme.

   Vielmehr ist unser Ziel, auch denjenigen APPLE-Besitzern "Tips und Tricks" mitzuteilen, die auf fertige (und oft teure) Programmangebote keinen Wert legen oder sich dies nicht leisten mögen.

4. Bei der Auswahl der Themenbereiche, die hier angesprochen werden, haben wir uns nicht von den Interessen spezieller Benutzerkreise leiten lassen: Es geht also nicht um spezielle kaufmännische Probleme oder nur um statistische Anwendungen oder um sonst ein spezielles, genau abgrenzbares Sachgebiet.

Vielmehr greifen wir unter programmlogischen und unter methodischen Gesichtspunkten unterschiedliche Programmierprobleme auf, die dann in den verschiedensten Anwendungsbereichen Verwendung finden können.

5. Bei der Vorstellung der einzelnen Programmtips und -tricks kommt es nicht auf möglichst hohe Eleganz der Programmierung an; auch die Rechengeschwindigkeit spielt für uns nicht die entscheidende Rolle.

   In der Hauptsache kommt es uns hingegen darauf an, die einzelnen Arbeitsschritte, die einen Programmablauf erzeugen, im Programm selbst noch erkennen zu lassen.

6. Aus dem vorangegangenen Punkt wird schon deutlich, daß wir unter "Tips und Tricks" nicht die letzten "ausgefeilten" Spezialitäten verstehen wollen, sondern Hilfestellungen, die auch für den Anfänger noch verständlich sein sollen (der "Könner" wird ohnehin auch ohne unsere Hilfestellungen sich mit den Besonderheiten und Spezialitäten des APPLE II vertraut machen können).

Zusammenfassend ist also festzuhalten, daß wir nicht nur Problemlösungen präsentieren, sondern insbesondere auch die <u>einzelnen Schritte</u> eines Problemlösungsweges aufzeigen wollen, damit dieser Problemlösungsweg nachvollziehbar wird.

Das wesentliche Problem beim Computereinsatz besteht nämlich nicht darin, ein Programm "zum Laufen" zu bringen, sondern vielmehr darin, die Struktur eines Problems zu erkennen, also ein Problem zu analysieren, seine Lösung "im Kopf" vorzubereiten, bevor ein Programm erstellt wird, das die Lösung dann dem Rechner überträgt.

Auch diese Zielsetzung legt eine Beschränkung auf einige typische Problemstellungen und Beispiele nahe und den Verzicht darauf, optimale Programmstrukturen zugunsten nachvollziehbarer Programme vorzustellen.

Dies bedeutet, daß die einzelnen Programme, die in den späteren Kapiteln benutzt werden, in der Regel nach folgendem Schema präsentiert werden:

1. Vorstellung des jeweiligen Problems;
2. Entwicklung des adäquaten Computerprogramms;
3. Programmbeschreibung und gegebenenfalls Vorstellung der Ergebnisse.

Erst seit kurzer Zeit ist ein neuer Rechner der APPLE-II-Gruppe auf dem Markt, der APPLE IIc. Es handelt sich um eine Weiterentwicklung, die mit 128K standardmäßig einen größeren Speicher hat als seine Vorgänger. Wesentlich für den Leser dieses Buches ist vor allem die Feststellung, daß der APPLE IIc kein völlig neuer Rechner ist.

Zwar wird er mit dem Betriebssystem ProDOS geliefert, aber auch das "alte" DOS 3.3 steht noch zur Verfügung. Das bedeutet, daß alle Programme, die für dieses Buch auf der Grundlage von DOS 3.3 entwickelt wurden, auf dem neuen Rechner gestartet werden können, auch wenn sie Anweisungen für den direkten Speicherzugriff und den Aufruf von Maschinenprogrammen enthalten. Außerdem können DOS 3.3-Dateien mit Hilfe einer mitgelieferten Systemdienstprogrammdiskette umgewandelt werden in ProDOS-Dateien; die Dateinamen können sich dadurch allerdings verändern. ProDOS läßt nur 15 Zeichen für die Namen zu, Leerzeichen, Umlaute und ß sind nicht zulässig. Blanks werden bei der Umwandlung in Punkte übersetzt.
Es war nicht immer möglich, alle Feinheiten des neuen Betriebssystems mit in dieses Buch aufzunehmen; so sei nur in allgemeiner Form darauf hingewiesen, daß APPLESOFT-BASIC das gleiche geblieben ist, aber bei Einsatz von 2 Diskettenlaufwerken gibt es Unterschiede. Der APPLE IIc hat ein eingebautes Laufwerk, der Aufruf eines zweiten, externen Laufwerks geht über PR#7. Dies entspricht der üblichen Slot-Schreibweise, obwohl der IIc keine Steckplätze mehr hat. Unter dem Betriebssystem ProDOS entfällt der Befehl INIT, dafür gibt es ein Systemdienstprogramm, das Disketten formatiert (für 3 verschiedene Betriebssysteme), und ein weiteres, mit dem zwei Systemdateien auf eine neue Diskette kopiert werden müssen, damit sie gebootet werden kann.

Vorsicht sollte der Leser walten lassen beim Übertragen des Kapitels 6 auf den neuen Rechner, insbesondere beim Einsatz des PROGRAM LINE EDITOR. Ähnliches gilt für den Abschnitt "Initialisierung" des Kapitels 7, der für Benutzer des Betriebssystems ProDOS uninteressant ist. Allerdings ist es auch unter ProDOS möglich, mit dem Booten einer Diskette ein Programm zu starten (ähnlich dem Initialisierungsprogramm); dieses muß den Namen STARTUP haben.

Abschließend sei daran erinnert, daß dieses Buch von zwei Verfassern geschrieben wurde. Gewisse - vor allem stilistische - Unterschiede ließen sich deshalb nicht ganz vermeiden.

Kapitel 1: Grundlegende BASIC-Elemente

1.1 Einführung

In diesem Kapitel soll zunächst ein kurzer Überblick über die wichtigsten und grundlegenden BASIC-Anweisungen geboten werden. Wir betrachten dieses Kapitel also quasi als Wiederholung oder als Auffrischung schon vorhandener Programmierkenntnisse. Aber auch der Anfänger wird so in die Lage versetzt, die Ausführungen der folgenden Kapitel zu verstehen. Für ihn mag allerdings dieser knappe BASIC-Wiederholungskurs zu kurz geraten sein: Vieles aber wird in den folgenden Kapiteln, im Zusammenhang mit der Erstellung konkreter BASIC-Programme, klar werden. Was hier also zu knapp erscheint, erfährt die ausführlichere Erläuterung im Rahmen der einzelnen Programmbeispiele der späteren Kapitel.

Wie auch schon in den Vorbemerkungen, sei hier ebenfalls darauf hingewiesen, daß Beschränkungen erforderlich sind: Nicht alles, was mit der Programmiersprache BASIC beim Einsatz der APPLE II-Rechner möglich ist, soll hier besprochen werden, sondern nur die wichtigsten und am häufigsten benutzten Anweisungen.
Ergänzend werden einige spezielle Anweisungen dann in den späteren Kapiteln - jeweils einleitend - kurz vorgestellt, wenn sie für bestimmte Programme tatsächlich benötigt werden.

1.2 BASIC

Wenn ein Rechner für uns ein Problem lösen soll, so benötigt er dazu eine Folge von Anweisungen.
Eine solche Folge nennt man ein Programm.

Ein solches Programm muß in einer Sprache geschrieben werden, die ein Rechner verstehen kann.
Die heute am häufigsten bei Kleinrechnern eingesetzte Sprache ist BASIC.

BASIC	=	Beginner's All-Purpose Symbolic Instruction Code

Diese Sprache ist allerdings nicht genormt, sondern es existieren unterschiedliche BASIC-Dialekte.
Diese überschneiden sich allerdings in einer recht großen gemeinsamen Schnittmenge von grundlegenden Sprachelementen; aber gerade der APPLE II arbeitet mit einer BASIC-Version (Applesoft BASIC), welche einige spezielle Besonderheiten aufweist, die wir uns zunutze machen können.

Bevor nun die einzelnen Sprachelemente besprochen werden, muß auf einen wichtigen Sachverhalt aufmerksam gemacht werden:

Jede Aufgabe, die ein Rechner erledigen soll, macht die Eingabe von drei Gruppen von Informationen erforderlich:

1. Daten
2. Programmanweisungen (auch Statements genannt)
3. Kommandos (Mitteilungen an das Betriebssystem)

Daten können Ziffern, Zahlen, Buchstaben, Worte, Texte oder sonstige Informationen sein.

Die Programmanweisungen erzeugen die notwendige Folge von Anweisungen, die zur gewünschten Verarbeitung der Daten benötigt werden. Diese Anweisungen werden in BASIC-Programmen mit Satznummern durchnumeriert (üblicherweise in Zehnerschritten).

Die Kommandos sind Mitteilungen an das Betriebssystem des Rechners und betreffen deshalb insbesondere organisatorische Aufgaben des Datenverarbeitungsprozesses. Kommandos erhalten keine Satznummern.

1.3 Einfache Ausgabe- und Eingabeanweisungen

Zur Ausgabe von Informationen dient das PRINT-Statement, das in unterschiedlichen Varianten auftreten kann. Die folgenden Beispiele erläutern dies:

1Ø	PRINT 3	Ausdruck der Zahl 3 auf dem Bildschirm;
2Ø	PRINT 3+5	Ausdruck des Rechenergebnisses 8;
3Ø	PRINT "APPLE"	Ausdruck des Strings (=Zeichenfolge) APPLE;
4Ø	PRINT X	Ausdruck des Speicherfeldes, das den Namen X trägt;
5Ø	PRINT	Ausgabe einer leeren Zeile.

Die Möglichkeiten, die in den Statements 1Ø bis 4Ø genannt wurden, können in einem PRINT-Statement mehrfach und/oder gemischt auftreten. Sie sind dann durch Komma oder Strichpunkt zu trennen.

1Ø PRINT "SUMME=";3+5

Diese Anweisung gibt erst den String SUMME =, daneben das Rechenergebnis 8 an, also:

SUMME=8

2Ø PRINT 3,8

Ausgabe der Zahlen 3 und 8, also:

3 8

(Will man stattdessen die Dezimalzahl 3.8 benutzen, so muß in BASIC ein Dezimalpunkt verwendet werden).

Wird der Strichpunkt als Trenner benutzt, so werden die Teilausdrucke aneinandergerückt, die Verwendung des Kommas hingegen rückt die Ausdrucke auseinander.

Jede PRINT-Anweisung bewirkt den Ausdruck einer neuen Zeile, es sei denn, diese Anweisung endet mit einem Komma oder einem Strichpunkt.

Das Wort PRINT kann abgekürzt werden durch ?.

Weiterhin gilt, daß mehrere BASIC-Anweisungen, so auch z.B. PRINT-Anweisungen, in einem Satz stehen dürfen. Sie sind dann durch Doppelpunkte voneinander zu trennen.

```
1Ø  ?3+5
2Ø  ?
3Ø  ?11-2
```

Diese Anweisungsfolge führt zu dem Ausdruck

```
8
9
```

Diese Anweisungsfolge kann auch, wie oben beschrieben, in einem Satz zusammengefaßt werden:

```
1Ø  ?3+5:?:?11-2
```

Sind Verschiebungen in der Ausgabezeile erwünscht, so ist dies mit der TAB-Funktion möglich:

1Ø	?TAB(5)3	Ausgabe der Zahl 3 fünf "Anschläge" rechts vom linken Bildschirmrand.

Zur Eingabe von Informationen stehen zunächst das LET- und das INPUT-Statement zur Verfügung.

1Ø	LET X=3.2	Das Feld mit dem Namen X, welches das Betriebssystem selbstätig bereitstellt, wird mit dem Wert 3.2 belegt.
2Ø	LET Y=2*X	Das Feld Y wird mit dem Wert 6.4 belegt.
3Ø	LET Y=Y+1	Das Feld Y wird jetzt mit dem Wert 7.4 belegt.
4Ø	LET N$="OTTO"	Das Feld N$ wird mit dem String OTTO belegt.

Es ist zu beachten, daß der Variablenname, der im LET-Statement links vom Gleichheitszeichen steht, mit einem Buchstaben beginnen muß.
Wird ein $ angehängt, steht das entsprechende Feld für Strings zur Verfügung, die bei der Eingabe in Anführungszeichen einzuschließen sind.

Das Wort LET darf im LET-Statement entfallen:

1Ø	XX=(3+5)/2	Das Feld XX wird mit dem Wert 4 belegt.
2Ø	Y2=XX↑2	Das Feld Y2 wird mit dem Wert 16 belegt.

Das LET-Statement eignet sich also besonders gut zum Durchführen von Rechenaufgaben. Dabei können eine Reihe von Funktionen nutzbringend verwendet werden.

```
1Ø  LET X=SQR(16)
2Ø  ?X
```

Berechnet wird die Wurzel aus 16 (=4); dieser Wert wird ausgegeben.

Weitere wichtige Funktionen sind:

SIN, COS, TAN	: Winkelfunktionen
INT	: Bestimmt den größten, ganzzahligen Teil in einer Zahl
ABS	: Absolutwert
LOG	: Natürlicher Logarithmus
EXP	: Exponentialfunktion
RND	: Bestimmt eine Zufallszahl

Neben dem LET-Statement steht zur Informationseingabe das INPUT-Statement zur Verfügung:

```
1Ø  INPUT X
```

: Anforderung eines Werts für die Variable X während der Programmabarbeitung

```
2Ø  INPUT A,B,C
```

: Anforderung von drei Werten

```
3Ø  INPUT A$,B$
```

: Anforderung von zwei Strings

```
4Ø  INPUT "BITTE EINE ZAHL:";X
```

wie 1Ø, aber mit zusätzlicher Textausgabe

```
5Ø  INPUT "BITTE VOR-UND NACHNAME:";V$,N$
```

wie 3Ø, aber mit zusätzlicher Textausgabe.

Beim INPUT-Statement ist darauf zu achten, daß dem Rechner genau so viele Werte (Zahlen oder Strings - je nach Variablentypen) gegeben werden, wie Variablen in der Variablenliste hinter dem Schlüsselwort INPUT aufgelistet wurden. Diese Variablennamen sind durch Komma voneinander zu trennen, ebenso die einzugebenden Werte.

Jedes BASIC-Programm ist sinnvollerweise mit dem Statement END abzuschließen.

Kommentare, Anmerkungen, Überschriften und dergl. können mit dem REM-Statement eingefügt werden.

Beispiel:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 1,1)"1.1 - PYTHAGORAS"
60  PRINT : PRINT : PRINT : PRINT : PRINT
70  INPUT "ERSTE KATHETE  : ";A
80  INPUT "ZWEITE KATHETE  : ";B
90 C =  SQR (A ^ 2 + B ^ 2)
100  PRINT : PRINT : PRINT "HYPOTHENUSE      = ";C
110  PRINT : PRINT "ENDE"
120  END
```

1.4 Erste Kommandos

Ein BASIC-Programm, wie das eben vorgestellte, wird vom Rechner erst dann abgearbeitet, wenn wir dem Betriebssystem das entsprechende Kommando geben.
Es ist dies das Kommando

RUN

Wenn wir z.B. nach einem Programmlauf das Programm, welches vielleicht über den oberen Bildschirmrand hinausgewandert ist, wieder sehen wollen, benutzen wir das Kommando

LIST

Das Kommando LIST kann sich auch auf einzelne Sätze oder Satzgruppen beziehen, wie folgende Beispiele zeigen:

LIST 5Ø-15Ø	: Sätze 5Ø-15Ø werden gelistet
LIST -8Ø	: Alle Sätze bis 8Ø werden gelistet
LIST 1ØØ-	: Alle Sätze ab 1ØØ werden gelistet
LIST 6Ø	: Satz 6Ø wird gelistet.

Soll der Inhalt des Programmspeichers gelöscht werden - z.B. vor der Aufnahme eines neuen Programms -, so ist das Kommando

NEW

zu verwenden.

1.5 Programmverzweigungen

Häufig ist es sinnvoll, daß bestimmte Programmteile mehrfach durchlaufen werden. Um dies zu ermöglichen, benötigt man sog. Sprunganweisungen (oder Verzweigungen), wobei zunächst zwischen unbedingten und bedingten Sprüngen zu unterscheiden ist.

Der unbedingte Sprung wird mit dem Statement GOTO erreicht, wie folgendes Beispiel zeigt:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 10)"1.2 - ENDLOSSCHLEIFE"
60  PRINT : PRINT : PRINT : PRINT : PRINT
70 I = 1
80 Q = I * I
90  PRINT I,Q
100 I = I + 1
110  GOTO 80
120  END
```

Dieses Programm druckt alle natürlichen Zahlen und ihre Quadratzahlen, und zwar so lange, wie die Speicherkapazitäten des Rechners dies erlauben (also sehr lange!).

Zu lang dauernde Programmläufe (oder auch Auflistungen) kann man stoppen mit der Tastenkombination

CTRL und C

Will man mit der Programmabarbeitung dann an der Unterbrechungsstelle (eventuell) fortfahren, so ist das Kommando

CONT

einzugeben.

Statt das Programmabarbeiten (oder das Programm-Auflisten) zu unterbrechen, kann es auch "gebremst" werden, und zwar mit der Tastenkombination

CTRL und S

Zur Fortsetzung braucht dann nur irgendeine Taste bestätigt werden.

Der bedingte Sprung, (das IF...THEN-Statement), unterscheidet sich vom eben besprochenen unbedingten Sprung dadurch, daß er nur dann ausgeführt wird, wenn eine vorgegebene Bedingung erfüllt ist.

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 9)"1.3 - BEDINGTER SPRUNG"
60  PRINT : PRINT
70 I = 1
80 Q = I * I: PRINT I,Q:I = I + 1
90  IF I <  = 12 THEN  GOTO 80
100  PRINT "ENDE": END
```

In diesem Programm werden natürliche Zahlen (und ihre Quadrate) nur zwischen I=1 und I=2Ø ausgegeben. Danach wird das Programm beendet.

Allgemein gilt: Ist die logische Bedingung hinter dem Schlüsselwort IF erfüllt, so befolgt der Rechner die Anweisung, die hinter dem THEN folgt (oben: Sprung nach 3Ø). Es können hier auch mehrere Anweisungen folgen, die dann wieder durch Doppelpunkte voneinander zu trennen sind.
Ist die Bedingung hingegen nicht erfüllt, fährt die Programmabarbeitung beim unter dem IF folgenden Satz fort.

Bei manchen Aufgabenstellungen ist es auch sinnvoll, bestimmte Programmteile ganz "auszulagern" (sog. Unterprogramme oder subroutines zu bilden), um diese bei Bedarf "anzuspringen".
Dieser Ansprung erfolgt mit dem Statement GOSUB.
Der Rücksprung erfolgt dann mit dem Statement RETURN an die Stelle des Hauptprogramms, die nach dem GOSUB folgt.

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 10)"1.4 - UNTERPROGRAMM"
60  PRINT : PRINT : PRINT : PRINT
70  GOSUB 1000: REM  EINGABE
80 I = 1: GOSUB 2000: REM  DURCHSCHNITT
90  GOSUB 1000
100 I = I + 1: GOSUB 2000
110  REM  ETC.
998  PRINT : PRINT : PRINT "ENDE"
999  END
1000  REM  UP EINGABE
1010  PRINT : PRINT : PRINT
1020  INPUT "BITTE DREI WERTE : ";X,Y,Z
1030  RETURN
2000  REM  UP DURCHSCHNITT
2010 S = X + Y + Z
2020 AM = S / 3
2025  PRINT : PRINT : PRINT
2030  PRINT I;" .MITTELWERT : ";AM
2040  RETURN
```

1.6 Programmschleifen

Den mehrfachen Durchlauf bestimmter Programmteile kann man sehr einfach auch mit den Statements FOR... TO und NEXT erreichen, wie das folgende Beispiel illustriert:

Beispiel:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 12)"1.5 - SCHLEIFE"
60  PRINT : PRINT
70  FOR I = 1 TO 12
80  PRINT I,I * I
90  NEXT I
100  PRINT : PRINT "ENDE": END
```

Auch hier werden die natürlichen Zahlen von 1 bis 2Ø und ihre Quadratzahlen ausgegeben.

Zusätzlich kann das FOR...TO-Statement mit einer Schrittweite versehen werden, wenn man von der Schrittweite 1 abweichen möchte:

```
1Ø  FOR I=1 TO 1Ø STEP Ø.4
```

1.7 Dateneingabe

Die Informationseingabe über das LET- oder das INPUT-Statement ist für große Datenbestände umständlich. Besser dafür geeignet sind das DATA- im Zusammenwirken mit dem READ-Statement, insbesondere dann, wenn zusätzlich das DIM-Statement verwendet wird.

Mit dem DIM-Statement werden für den (oder die) genannten Variablennamen mehrere Speicherstellen von vornherein freigehalten:

```
1Ø  DIM X(2Ø),Y(15)
```

: Für X werden 21, für Y 16 Speicherstellen reserviert.

Mit dem DATA-Statement werden Daten bereitgestellt, mit dem READ-Statement werden sie durch den Rechner gelesen.

Dazu folgendes Beispiel:

```
30   PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40   PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50   PRINT : PRINT  TAB( 13)"1.6 - DATEN"
60   PRINT : PRINT
70   DIM X(15)
80   DATA  88,66,77,75,63,69,84,87,72,79,75,78,74,91,83
90   FOR I = 1 TO 15: READ X(I): NEXT I
100   PRINT "KONTROLLE : ": PRINT
110   FOR I = 1 TO 15: PRINT X(I);"  ";: NEXT I: PRINT : PRINT

120   REM  WEITERE DATENVERARBEITUNG
130   PRINT : PRINT : PRINT "HIER KANN NUN DIE WEITERE DATENVERAR-"
140   PRINT "BEITUNG ANGESCHLOSSEN WERDEN."
150   PRINT : PRINT : PRINT "ENDE": END
```

In dieser Anweisungsfolge nennt man X(I) eine <u>indizierte Variable;</u> die einzelnen gelesenen Werte sind unter den Namen X(Ø), X(1),... ansprechbar.

Variablen können auch mehrfach indiziert werden. Im folgenden Beispiel wird eine doppelte Indizierung verwendet:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 12)"1.7 - SCHACHTEL"
60  PRINT : PRINT
70  DATA  172,68
80  DATA  183,85
90  DATA  180,81
100  DIM X(3,2)
110  FOR I = 1 TO 3
120  FOR J = 1 TO 2
130  READ X(I,J)
140  NEXT J
150  NEXT I
160  PRINT : PRINT "KONTROLLE : ": PRINT
170  FOR I = 1 TO 3
180  FOR J = 1 TO 2: PRINT X(I,J);"  ";: NEXT J: PRINT : PRINT

190  NEXT I
200  PRINT : PRINT "HIER KOENNEN NUN WEITERE DATENVERARBEI-"
210  PRINT "TUNGSSCHRITTE ANGESCHLOSSEN WERDEN."
220  PRINT : PRINT "ENDE": END
```

In obigem Beispiel werden 4+3=12 Speicherstellen reserviert. Die einzelnen Felder heißen X(Ø,Ø),X(Ø,1), X(Ø,2),X(1,Ø),...
Der erste Index ist der sog. Zeilenindex, der zweite der Spaltenindex.

Es ist dabei zu beachten, daß per READ nicht mehr Werte gelesen werden können, als in den DATA-Statements bereitstehen, wohl aber weniger.
Eventuell folgende READ-Anweisungen fahren in der DATA-Liste dort fort zu lesen, wo die vorhergehende aufgehört hat.
Soll ein Datenbestand durch folgende READ-Statements wieder von Anfang an gelesen werden, so ist vorher das Statement RESTORE zu benutzen.

1.8 Stringbearbeitungsfunktionen

Ein "String" ist eine Zeichenfolge oder Zeichenkette, in der Buchstaben, Ziffern oder Sonderzeichen stehen können. Derartige Strings werden in BASIC in Anführungszeichen eingeschlossen.

Mit Hilfe der logischen Vergleichsoperatoren (<, >, =, <=, >=) können Strings hinsichtlich ihrer alphabetischen Aufeinanderfolge miteinander verglichen werden.

Mit Hilfe des Pluszeichens (+) können einzelne Strings zu neuen Gesamtstrings verkettet werden.

Es stehen darüber hinaus eine Reihe von Funktionen zur Verfügung, mit deren Hilfe Strings be- und verarbeitet werden können. Diese Funktionen sollen hier kurz vorgestellt werden, wobei auf Anwendungsbeispiele verzichtet wird (sie finden sich zur Genüge in den späteren Kapiteln).

LEN : Diese Funktion bestimmt die Länge eines Strings, also die Anzahl der Zeichen in der Zeichenkette.

Beispiel: L=LEN("HALLO") ergibt im Feld L den Wert 5.

VAL : Diese Funktion spaltet führende Ziffern eines Strings ab.

Beispiel: V=VAL("46ØØ DORTMUND") ergibt im Feld V den Wert 46ØØ.

STR$: Diese Funktion verwandelt eine Folge von Ziffern in einen String.

Beispiel: A$=STR$(46ØØ) ergibt in Feld A$ den String "46ØØ".

LEFT$: Diese Funktion spaltet von einem String, der als erstes Argument zu nennen ist, so viele Zeichen von links beginnend als Teilstring ab, wie im zweiten Argument verlangt wird.

Beispiel: B$=LEFT$("OSTERN",3) ergibt in Feld B$ den String "OST".

RIGHT$: Entsprechend von rechts beginnend.

Beispiel: C$=RIGHT$("W.VOSS",4) ergibt in Feld C$ den String "VOSS".

MID$: Diese Funktion hat drei Argumente:

1. Den zu behandelnden String;
2. Nummer des Symbols, bei dem die Abspaltung beginnen soll;
3. Zahl der Symbole, die abgespalten werden.

Beispiel: D$=MID$("HALLO",3,2) ergibt in Feld D$ den String "LL".

ASC : Diese Funktion bestimmt die Codeziffer des ASCII-Code für das Zeichen, welches als Argument genannt wird.

Beispiel: X=ASC("A") ergibt den Wert 65 in Feld X.

CHR$: Diese Funktion ermittelt zu einer als Argument eingegebenen Codezahl (s.o.) das entsprechende Symbol.

Beispiel: A$=CHR$(66) ergibt in Feld A$ den String "B".

1.9 Benutzung von Disketten

Disketten eignen sich hervorragend dazu, Programme und/oder Datenbestände auf Dauer zu speichern.

"Fabrikneue" Disketten müssen allerdings zunächst initialisiert (oder formatiert) werden. Man geht dabei folgendermaßen vor:

1. Einlegen der Betriebssystemdiskette in das Laufwerk (DOS-Master-Diskette);
2. Einschalten des Rechners;
3. Einlegen der neuen Diskette;
4. Eingabe z.B. folgenden Programms:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT, TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 9)"1.8 - INITIALISIERUNG"
60  PRINT : PRINT
70  REM  ************************
80  REM  INHALT DES INITIALISIERUNGSPROGRAMMS
90  PRINT "INHALT DES INITIALISIERUNGSPROGRAMMS :": PRINT :
PRINT
100  PRINT "DISKETTE NR. 1"
110  PRINT "VOSS, MAI 1984"
120  END
```

5. Eingabe des Kommandos

   INIT Name

   An der Stelle Name muß ein Programmname, z.B. P1 oder dergl., eingegeben werden.

Anmerkung: Jedes Programm, das auf einer Diskette gespeichert wird, muß einen Namen erhalten. Wir nennen ihn den Datei-Namen.

! Achtung : Während die Lampe brennt, darf die Laufwerkstür nicht geöffnet werden!

Auf die nun initialisierte Diskette können Programme aus dem Programmspeicher übertragen werden.
Dazu dient das Kommando SAVE.

Beispiel:

SAVE PROGR1

Soll entsprechend dieses Programm später einmal von der Diskette gelesen werden, benötigen wir:

LOAD PROGR1

Wollen wir wissen, welche Dateien auf einer Diskette gespeichert sind, so erhalten wir ein "Inhaltsverzeichnis" mit dem Kommando

CATALOG

Soll eine bestimmte Datei gelöscht werden, so geben wir ein:

DELETE Dateiname

Manchmal ist es auch sinnvoll, bestimmte Dateien umzubenennen:

RENAME alter Name, neuer Name

1.1o Ergänzungen

Erzeugt ein Programm sehr viele Ergebnisse, so wandern diese über den oberen Bildschirmrand hinaus. Dies kann man verhindern, wenn man an passenden Programmpositionen dafür sorgt, daß es anhält. Dazu dient die Anweisung

STOP

Ein per STOP-Anweisung angehaltenes Programm kann mit dem schon genannten Kommando CONT wieder in Gang gesetzt werden.

Beispiel:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 15)"1.9 - STOP"
60  PRINT : PRINT
70  FOR I = 1 TO 35
80 Q = I * I: PRINT I,Q
90  IF I / 10 =  INT (I / 10) THEN  PRINT "BITTE CONT EINGEBEN":
STOP
100  NEXT I
110  PRINT "ENDE": END
```

In ähnlicher Weise kann z.B. die GET-Anweisung verwendet werden. Wird sie erreicht, so wird die Programmabarbeitung eingestellt und der Rechner erwartet eine Tasteneingabe.

Beispiel:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 13)"1.10 - WARTEN"
60  PRINT : PRINT : PRINT "ICH WARTE.": PRINT : PRINT "BITTE
EINE TASTE DRUECKEN !": PRINT : PRINT : PRINT
70  GET A$: IF A$ = "" THEN 70
80  PRINT : PRINT : PRINT "TASTE ";A$;" WURDE GEDRUECKT."
90  PRINT : PRINT "JETZT KANN ES WEITERGEHEN."
100  REM  WEITERES PROGRAMM
110  PRINT : PRINT "ENDE": END
```

Schließlich soll auf die HOME-Anweisung hingewiesen werden. Sie bewirkt, daß der Cursor nach links oben geschickt (HOME-Position) und der Bildschirm gelöscht wird. Vor den Ergebnisdarstellungen ist dies ein recht nützlicher Schritt.

Ähnliche Wirkung haben die Anweisungen VTAB und HTAB:

VTAB Z bewirkt, daß das folgende PRINT zur Ausgabe in Zeile Z führt; HTAB S bewirkt Ausgabe in Spalte S.

1.11 Abschlußbeispiel

Mit einem umfangreicheren Beispiel soll dieses Kapitel beschlossen werden, um das Zusammenwirken der wichtigsten BASIC-Sprachelemente zusammenfassend zu illustrieren.

Aufgabenstellung

Es soll ein BASIC-Programm entwickelt werden, welches einen beliebigen Datenbestand der Größe nach sortiert.

Problemanalyse

Der Rechner kann nur zur Lösung solcher Probleme eingesetzt werden, die wir im Kopf quasi schon gelöst haben. Das soll heißen, daß uns der Lösungsweg klar sein muß, bevor ein problemlösendes Programm geschrieben werden kann.

Diese notwendige Vorstrukturierung des Problems nennt man Problemanalyse. Sie erstreckt sich bei dieser Aufgabenstellung insbesondere auf die Frage, wie der eigentliche Sortierprozeß, also der "Kern" des zu entwickelnden Programms ablaufen muß. Am Beispiel eines "Spieldatenbestands" kann diese Analyse vorgeführt werden.

"Sortieren" bedeutet, daß wir je zwei Werte aus dem Datenbestand miteinander vergleichen:

Wir vergleichen den ersten Wert Schritt für Schritt mit allen übrigen. Ist dabei der erste Wert größer als einer der Vergleichswerte, so müssen beide miteinander vertauscht werden.

Dadurch wird erreicht, daß dann, wenn der erste Wert mit allen übrigen verglichen wurde (nach der 1. Runde), an der ersten Stelle nun der insgesamt kleinste Wert steht.

Dann vergleichen wir den jetzt an zweiter Stelle stehenden Wert mit allen übrigen, außer mit dem ersten Wert. Ist der zweite Wert größer als einer der Vergleichswerte, so wird wieder getauscht.

Dadurch wird erreicht, daß am Schluß dieser zweiten Vergleichsrunde an der 2. Stelle der zweitkleinste Wert steht.
Dann vergleichen wir den jetzt an dritter Stelle stehenden Wert mit allen übrigen, außer mit denen, die schon auf der ersten und auf der zweiten Stelle stehen. Am Ende dieser dritten Runde haben wir den drittkleinsten Wert gefunden und auf die dritte Stelle gesetzt.

Auf diese dritte Runde folgen so lange weitere Vergleichsrunden wie Vergleiche möglich sind:

Haben wir beispielsweise vier Werte, so müssen drei derartige Runden durchlaufen werden. In der ersten Runde gibt es drei Vergleiche, in der zweiten Runde zwei Vergleiche und in der dritten Runde noch einen Vergleich.

Schematisch sieht dies folgendermaßen aus:

1. Runde:	Vergleich 1:	Feld 1 mit Feld 2
	Vergleich 2:	Feld 1 mit Feld 3
	Vergleich 3:	Feld 1 mit Feld 4
2. Runde:	Vergleich 1:	Feld 2 mit Feld 3
	Vergleich 2:	Wert 2 mit Feld 4
3. Runde:	Vergleich 1:	Feld 3 mit Feld 4

Es ist dabei zu beachten, daß die Inhalte der Felder immer andere sein können, je nachdem, ob und an welchen Stellen getauscht wurde.

Programmablauf

Den Programmablauf kann man sich am besten verdeutlichen, wenn man sich "Spieldaten" vorgibt:
Zu sortieren sei beispielsweise der folgende Datenbestand, der fünf Daten umfaßt:

6	3	5	9	7

Die fünf Positionen, in denen sich diese fünf Werte befinden (die fünf "Kästchen" oben), sollen im folgenden als Felder bezeichnet werden.

1. Runde (I=1)

	vorher	nachher
Vergleich 1 (J=2) (Feld 1 mit Feld 2)	6 3 5 9 7	3 6 5 9 7
Vergleich 2 (J=3) (Feld 1 mit Feld 3)	3 6 5 9 7	3 6 5 9 7
Vergleich 3 (J=4) (Feld 1 mit Feld 4)	3 6 5 9 7	3 6 5 9 7
Vergleich 4 (J=5) (Feld 1 mit Feld 5)	3 6 5 9 7	3 6 5 9 7

Ausdruck X(1)=3

2. Runde (I=2)

Vergleich 1 (J=3) (Feld 2 mit Feld 3)	3 6 5 9 7	3 5 6 9 7
Vergleich 2 (J=4) (Feld 2 mit Feld 4)	3 5 6 9 7	3 5 6 9 7
Vergleich 3 (J=5) (Feld 2 mit Feld 5)	3 5 6 9 7	3 5 6 9 7

Ausdruck X(2)=5

3. Runde (I=3)

	vorher	nachher
Vergleich 1 (J=4) (Feld 3 mit Feld 4)	3 5 6 9 7	3 5 6 9 7
Vergleich 2 (J=5) (Feld 3 mit Feld 5)	3 5 6 9 7	3 5 6 9 7
		Ausdruck X(3)=6

4. Runde (I=4)
(= letzte Runde)

Vergleich 1 (J=5) (Feld 4 mit Feld 5)	3 5 6 9 7	3 5 6 7 9
		Ausdruck X(4)=7

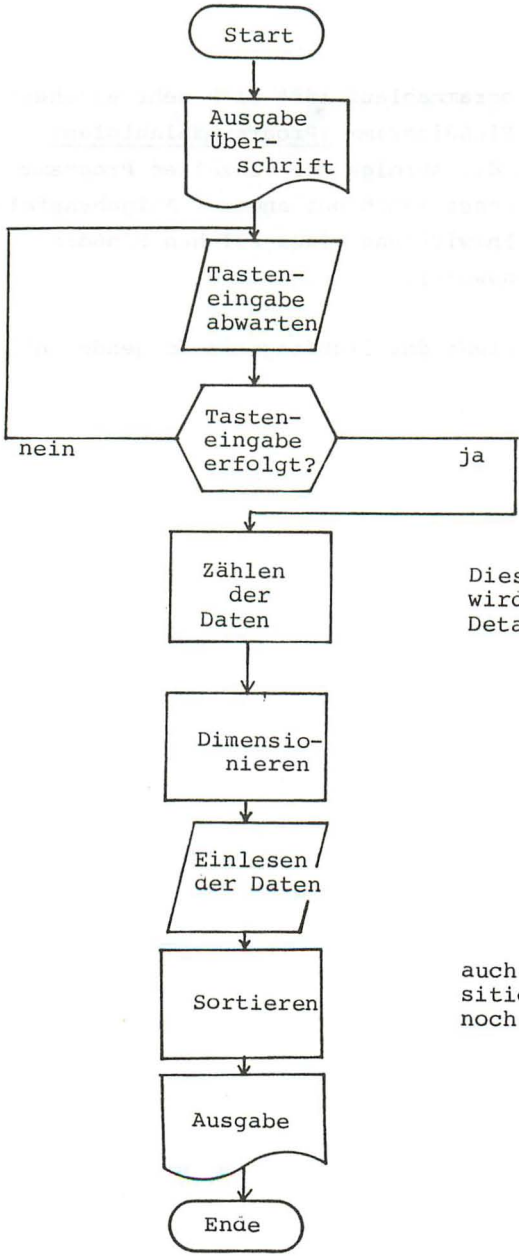
Beendigung der I-Schleife;

Ausdruck X(5)=9

Flußdiagramm

Der geplante Programmablauf läßt sich sehr anschaulich durch ein Flußdiagramm (Programmablaufplan) darstellen, das die Abfolge der einzelnen Programmschritte vorzeichnet (auch bei anderen Aufgabenstellungen ist die Entwicklung eines solchen Flußdiagramms empfehlenswert).

In diesem Fall sieht das Flußdiagramm folgendermaßen aus:

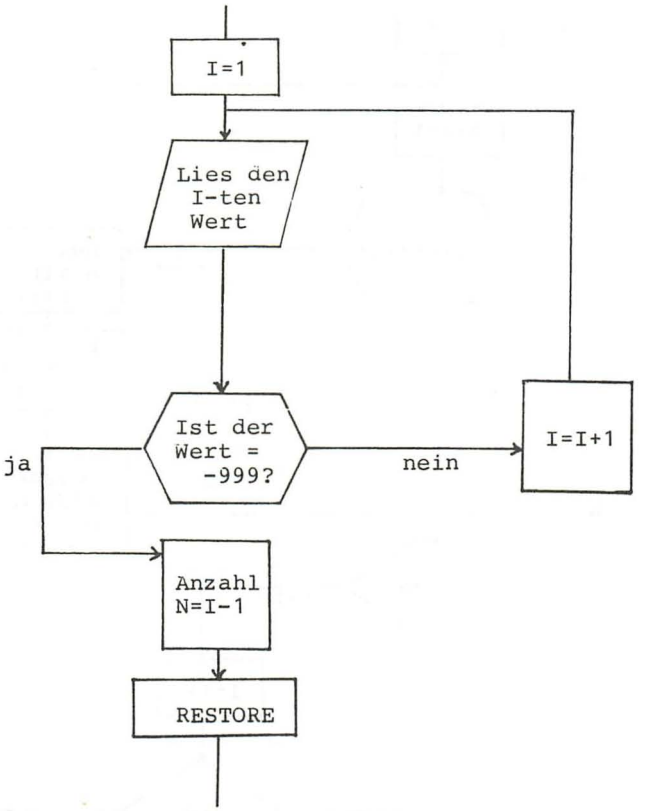


Diese Position wird noch im Detail erläutert;

auch diese Position wird noch aufgegliedert.

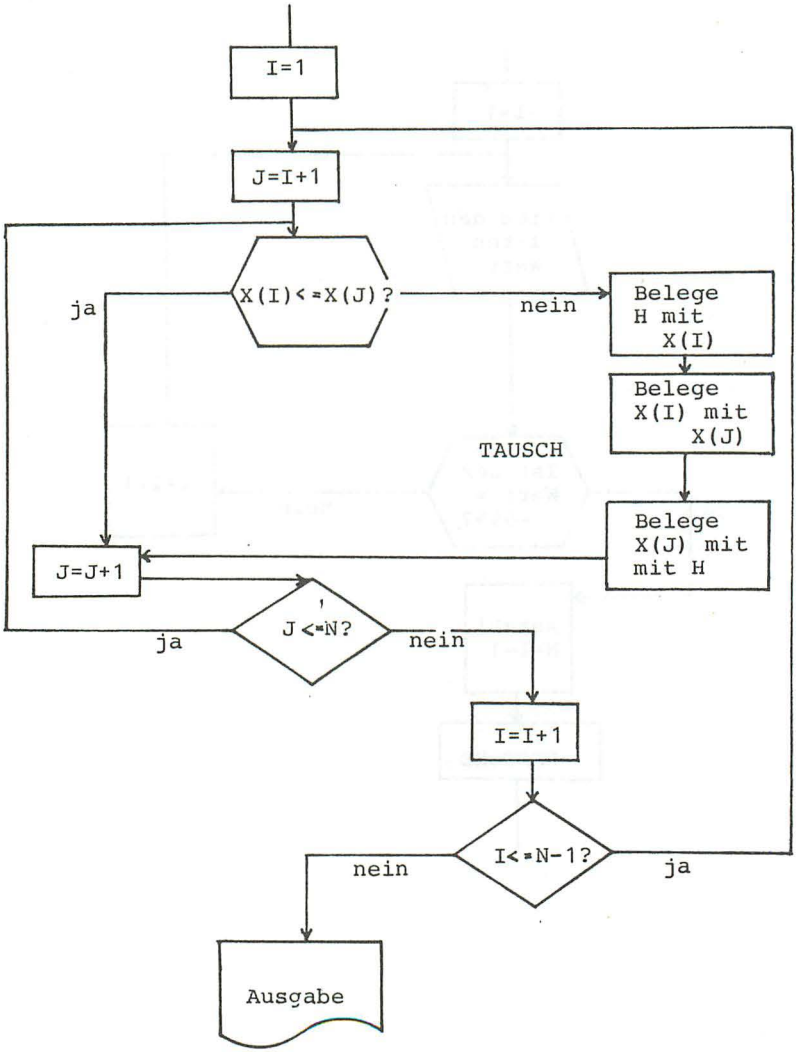
Teilproblem : <u>Zählen</u> von Daten

(die Zahl -999 markiere das Ende des Datenbestandes)



Teilproblem : <u>Sortieren</u>

(N = Anzahl der Werte)



Programm

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 11)"1.11 - SORTIEREN"
60  PRINT : PRINT : PRINT : PRINT
70  PRINT "DAS FOLGENDE PROGRAMM SORTIERT EINEN "
80  PRINT "BELIEBIGEN DATENBESTAND DER GROESSE"
90  PRINT "NACH.": PRINT
100  PRINT "DIE DATEN WERDEN UEBER DATA EINGEGEBEN"
110  PRINT "(STATEMENTS 500 FF.).": PRINT
120  PRINT "DIE LETZTE ANGABE MUSS EIN UNREALISTI-"
130  PRINT "SCHER WERT SEIN (HIER Z.B. -1)."
140  PRINT : PRINT : PRINT  TAB( 6)"BITTE EINE TASTE DRUECKEN
!"
150  GET A$: IF A$ = "" THEN 150
160  HOME
170  REM  ZAEHLEN DER DATEN
180  FOR I = 1 TO 9999: READ Z: IF Z =  - 1 THEN 200
190  NEXT I
200 N = I - 1
210  PRINT "ANZAHL DER WERTE = ";N
220  REM  LESEN DER DATEN
230  RESTORE
240  DIM X(N)
250  FOR I = 1 TO N: READ X(I): NEXT I
260  REM  KONTROLLAUSDRUCK
270  PRINT : PRINT : PRINT "KONTROLLE : ": PRINT
280  FOR I = 1 TO N: PRINT X(I);"  ";: NEXT I: PRINT
290  REM  SORTIEREN UND AUSGABE
295  PRINT : PRINT : PRINT
300  PRINT : PRINT "SORTIERT : ": PRINT
310  FOR I = 1 TO N - 1: FOR J = I + 1 TO N
320  IF X(I) < X(J) THEN 340
330 H = X(I):X(I) = X(J):X(J) = H
340  NEXT J: PRINT X(I);"  ";
350  NEXT I: PRINT X(I)
360  PRINT : PRINT : PRINT "ENDE DER AUSGABE": END
500  REM  DATEN
510  DATA  7,3,2,9,8,1,4
520  DATA  -1
530  END
```

Dieses Beispiel zeigt recht anschaulich, wie ein noch recht einfaches und überschaubares Problem so aufbereitet werden kann, daß es durch einen Computer erledigt werden kann.

Noch einmal sei daran erinnert, daß wir weder auf möglichst hohe "Programmeleganz" noch auf hohe Rechengeschwindigkeit Wert gelegt haben.
Dies wird auch in den folgenden Kapiteln nicht der Fall sein.

Kapitel 2: Die Nutzung peripherer Geräte

2.1 Vorbemerkungen

Wer sich einen Kleinrechner kauft, begnügt sich häufig zunächst - nicht zuletzt aus finanziellen Gründen - mit der Zentraleinheit selbst, die ja beim APPLE die Eingabetastatur mit einschließt, und dem Bildschirm als Ausgabeeinheit.

Nach kurzer Zeit aber wird man feststellen, daß als zusätzliches Ausgabegerät ein Drucker und als peripheres (externes) Speichergerät wenigstens ein Diskettenlaufwerk unentbehrlich sind.

Damit nämlich eröffnen sich vielfältige zusätzliche Möglichkeiten der Computernutzung, von denen einige hier angesprochen werden sollen.

2.2 Der Drucker

Schließt man an den Rechner beispielsweise einen Drucker an, so kann man Programmlisten und/oder Programmergebnisse "schwarz auf weiß nach Hause tragen".

Beim APPLE-Computersystem muß dazu die zu dem Drucker gehörende Drucker-Steuerkarte in den Slot 1 eingesteckt werden (Installationshinweise beachten!).

Entsprechende Steckkarten werden auch für andere peripheres Geräte notwendig. Man bezeichnet diese Steckkarten mit der dazugehörigen "Schnittstellen" (interfaces).

Beim Übergang von Informationen vom Rechner zu einem peripheren Gerät (oder umgekehrt) ist im Direktmodus das Kommando

PR# Zahl

zu benutzen, wobei an der Stelle Zahl die Nummer des jeweiligen Kartensteckplatzes einzufügen ist.

Im Programm-Modus ist die Ansteuerung des Gerätes etwas umständlicher:

Die Anweisung PR#1 (bezogen auf Slot 1) wird dann innerhalb einer PRINT-Anweisung verwendet, in der außerdem das Symbol aufgenommen wird, welches durch die Tastenkombination

CTRL und D

zustandekommt.

Dies ist ein Zeichen, das auf dem Bildschirm nicht erscheint.

Ein entsprechender Programmanfang könnte folgendermaßen aussehen:

```
1Ø  D$ =""                Zwischen den Anfüh-
                          rungszeichen steht
                          das nichtdruckende
                          Zeichen
                             CTRL-D
2Ø  ?D$;"PR # 1"
 :
```

Alle nun nachfolgenden PRINT-Anweisungen führen zum Drucker, wenn das Drucker-Interface in Slot 1 steckt.

Bevor wir an geeigneten Beispielen diese Anweisungen näher betrachten, ist zu unterscheiden, ob der Drukker zur Ausgabe von Programmen oder von Programmergebnissen benutzt werden soll. Im ersten Fall arbeiten wir im Kommando-Modus, im zweiten Fall im Programm-Modus.

2.2.1 Ausgabe von Programmen

Durch geeignete Kommandos, die oben schon vorgestellt wurden, werden Programme statt auf dem Bildschirm auf dem Drucker gelistet.

Diese Kommandofolge lautet:

```
PR# 1
LIST
PR# Ø
```

Es gibt hier also keine besonderen Schwierigkeiten oder Besonderheiten, die zu beachten wären.

2.2.2 Ausgabe von Programmergebnissen

Wie schon erwähnt, ist jetzt die Anweisung PR# 1, zusammen mit [CTRL] und [D] in eine PRINT-Anweisung aufzunehmen.

Das folgende Beispiel illustriert die Vorgehensweise:

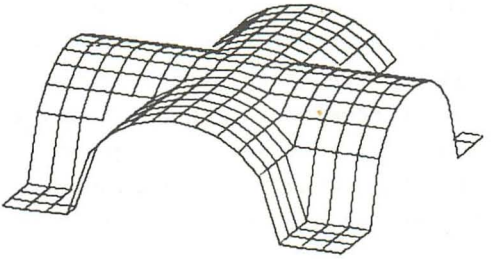
```
 30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
 40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
 50  PRINT : PRINT  TAB( 12)"2.1 - DRUCKER"
 60  PRINT : PRINT : PRINT : PRINT
 70 D$ = "": REM  CTRL UND D
 80  PRINT D$;"PR#1"
 90  PRINT "DIESER TEXT WIRD UEBER DEN"
 100  PRINT "DRUCKER AUSGEGEBEN."
 110  PRINT D$;"PR#0"
 120  END
```

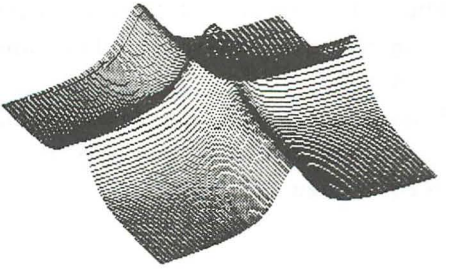
Nähere Erläuterungen dürften entbehrlich sein.

2.2.3 Graphiken

Die Graphiken, die wir mit dem APPLE erzeugen können (vergl. Kap. 4), sollen bei Bedarf auch über den Drukker festgehalten werden können.

Bei modernen Matrixdruckern ist dies kein Problem, da jeder einzelne Bildschirmpunkt auch einzeln darstellbar ist, so daß beispielsweise Bilder folgender Art möglich sind:





Allerdings ist die Programmierung solcher Druckerausgaben recht aufwendig - eigentlich so aufwendig, daß sie sich kaum lohnt.

Der Aufwand, der hier betrieben werden muß, darf nicht verwechselt werden mit der Erzeugung von Graphiken auf dem Bildschirm, die in Kap. 4 besprochen werden.
Dabei treten keine besonderen Probleme auf, weil es eine Reihe von Anweisungen gibt, die die Programmierung von Graphiken wesentlich erleichtern und ohne allzu großen Aufwand ermöglichen.

Anders hingegen ist es bei der Graphikausgabe auf dem Drucker: Hier müssen die einzelnen Positionen auf dem Druckerpapier, die bedruckt werden sollen - eine solche Druckgraphik ist ja letzten Endes nichts anderes als eine mehr oder weniger regelmäßige Anhäufung schwarzer Punkte - auf sehr umständliche Weise festgelegt werden. Man muß dabei mit hexadezimal verschlüsselten Adressen arbeiten, man muß die Speichereinteilung im Kopf haben usw.

Es empfiehlt sich deshalb, Hilfsprogramme oder Hardware-Ergänzungen einzusetzen, die diese organisatorischen Aufgaben erledigen.
Es steht für den APPLE eine "Parallel Drucker Interface Karte" zur Verfügung, die außerordentlich angenehme Eigenschaften hat:
Diese Karte erlaubt nämlich sowohl die Textausgabe wie auch eine komfortable Gestaltung der Ausgabe hochauflösender Graphik (vergl. dazu Kap. 4).

Die Karte ist in Steckplatz 1 einzustecken (Bedienungshinweise beachten!) und mit dem Drucker zu verbinden.

Kommandos, die sich an diese Karte richten, sind mit der CTRL -Taste einzuleiten.
Soll hingegen diese Karte während eines Programmlaufs, also nicht im Kommandomodus (Direktmodus) angesprochen werden, ist mit der CHR$-Funktion zu arbeiten.
Das voreingestellte Steuerwort ist CTRL - I; dem entspricht die Anweisung

```
PRINT   CHR$(9)
```

Die wesentlichen Anweisungen für die Ausgabe von Texten sind die folgenden:

PR# 1	:	Einschalten des Interface
PR# Ø	:	Ausschalten des Interface
CTRL-I8ØN	:	Druckerzeilenlänge wird auf 8Ø Zeichen eingestellt (es können alternative Zahlenwerte eingegeben werden). Wird ØØ eingegeben (an der Stelle, an der jetzt 8Ø steht), wird diese Funktion abgeschaltet. Als Programmzeile könnte diese Anweisung folgendermaßen aussehen: 1Ø PRINT CHR$(9);"8ØN"
CTRL-I I	:	Alle Zeichen werden auf dem Bildschirm <u>und</u> auf dem Drucker ausgegeben.

CTRL-I5L : Es wird ein linker Rand von 5 Zeilen gelassen (alternativ besetzbar).

CTRL-I7ØR : Ab der 7Ø. Spalte wird auf Leerzeichen geachtet. Wird ein Leerzeichen entdeckt, erfolgt ein Zeilenvorschub (der Wert 7Ø ist austauschbar).

CTRL-I6ØP : Es werden 6Ø Zeilen pro Seite ausgegeben (alternativ besetzbar), dann folgen automatisch 6 Zeilenvorschübe (abschalten durch ØØ).

CTRL-IS : Druckt den Textinhalt des Bildschirms.

Dazu ein einfaches Illustrationsbeispiel

```
 80  PR# 1
 90  PRINT  CHR$ (9);"50N"
 95  PRINT  CHR$ (9);"20L"
 97  PRINT  CHR$ (9);"I"
 100  PRINT "DIESES PROGRAMM ZEIGT
      , IN WELCHER WEISE DIE DRUCK
      ER-INTERFACE-KARTE,"
 110  PRINT "ÜBER DIE OBEN GESPROC
      HEN WURDE, BENUTZT WERDEN KA
      NN"
 120  PRINT : PRINT : PRINT : PRINT
      "ENDE": END
 130  PR# 0
```

Ergebnis :

```
DIESES PROGRAMM ZEIGT, IN WELC
 HER WEISE DIE DRUCKER-INTERFA
 CE-KARTE,
ÜBER DIE OBEN GESPROCHEN WURDE
 , BENUTZT WERDEN KANN


ENDE
```

Der Leser gebe nach Ablauf dieses Programms das Kommando:

CTRL – I S

Was geschieht?

Die wesentlichen Anweisungen für die Graphik-Ausgabe sind die folgenden:

Der Inhalt des Graphikspeichers des APPLE kann - wie schon erwähnt - über diese Interface-Karte direkt ausgegeben werden. Dazu benötigen wir lediglich die Anweisung

CTRL - I und "G" plus Argument

also z.B. als Kommando oder Programm-Statement:

```
(nn)  ?CHR$(9);"G Argument"
```

Als Argument kommt in Frage (diese Aufzählung ist nicht vollständig, sondern berücksichtigt nur die interessantesten Möglichkeiten):

2: Ausdruck der zweiten Bildschirmseite (Voreinstellung: erste Seite);

D: Ausgabe der Graphik in doppelter Größe;

I: Invertieren der Graphik (Voreinstellung: weiße Bildschirmpunkte werden schwarz gedruckt);

R: Drehung der Graphik um 9o°;

A: Verknüpfung (AND) beider Bildschirmseiten (s. Kap. 4 zur näheren Erläuterung).

Es ist bei diesen Anweisungen zu beachten, daß ein RESET notwendig wird, wenn der Drucker bei einer bestimmten Arbeit mit dem Platz nicht auskommt.

Auf ein Beispiel zur Benutzung dieser Anweisungen zur Graphikausgabe auf dem Drucker soll hier verzichtet werden. Wir kommen in Kap. 4 darauf zurück, wo es ausschließlich um die Graphikmöglichkeiten des APPLE geht.

2.3 Diskettenlaufwerk

2.3.1 Vorbemerkungen

Der wesentliche externe Speicher bei Mikrocomputersystemen sind die Disketten, die auf den dazugehörigen Diskettenlaufwerken beschrieben und gelesen werden können.

Im Gegensatz zu manchen anderen Computersystemen, haben die APPLE-Diskettenlaufwerke keinen eigenen Stromanschluß, sondern werden über die Stromversorgung des Rechners mit bedient. Dies bedeutet, daß beim Einschalten des Rechners auch das Laufwerk anspringt. Dabei werden - sofern eine initialisierte Diskette eingelegt ist - die für die Diskettenverwaltung notwendigen Teile des Betriebssystems geladen, so daß sofort mit der (oder den) Diskette(n) gearbeitet werden kann.

Man nennt diese Vorgehensweise den sog. "Kaltstart" des Systems.

Ist keine oder eine nichtinitialisierte Diskette eingelegt, so ist das angesprungene Laufwerk mit der RESET-Taste zu stoppen, bevor mit dem Rechner gearbeitet werden kann.

Das Diskettenlaufwerk kann für zwei verschiedene Aufgabenstellungen verwendet werden:

1. Zum Speichern und/oder Wiedereinlesen von Programmen;

2. Zur interaktiven Nutzung von Dateien (insbesondere von Datenbeständen), d.h. also Diskettenzugriff vom laufenden Programm aus.

Mit beiden Möglichkeiten wollen wir uns im folgenden kurz beschäftigen, bevor an einem ausführlichen Beispiel auf spezielle Tips beim Diskettenhandling eingegangen wird.

Es ist bei der Benutzung von Disketten wichtig, von vornherein zwei Arten der Informationsspeicherung zu unterscheiden:

1. Sequentielle Speicherung
2. Random access Speicherung

Die Erläuterung dieser Begriffe muß am Begriff der Datei ansetzen, was uns Gelegenheit bietet, einige Grundbegriffe der Datenverarbeitung kurz zu skizzieren:

Man stelle sich eine Fragebogenerhebung vor, bei der die Befragten zu folgenden Sachverhalten um Auskunft gebeten wurden:

1. Geschlecht
2. Geburtsjahr
3. Familienstand

:

etc.

Der sich ergebende Datenbestand läßt sich in Form einer Datenmatrix darstellen:

Person Nr.	Geschlecht	Geburtsjahr	Familienstand	...
ØØ1	männlich	1942	verheiratet	...
ØØ2	weiblich	1953	ledig	...
ØØ3	männlich	1963	ledig	...
ØØ4	männlich	1958	verheiratet	...
⋮				

Alle weiteren Datenverarbeitungsprozesse können an dieser Datenmatrix ansetzen, wobei es gleichgültig bleibt, ob wir die einzelnen Angaben wie vorliegend verwenden, oder ob sie, soweit dies möglich ist, in numerische Werte transformiert werden (z.B. "männlich" = Ø, "weiblich" = 1 usw.).

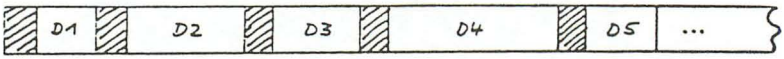
Jede Zeile einer solchen Matrix heißt Datensatz (kurz: Satz), bzw. record, set oder case.
Jede Spalte heißt array. Der Kreuzungsbereich von einer Spalte und einer Zeile heißt Feld (field).

In einem Feld finden wir also die Ausprägung einer Variablen für einen Merkmalsträger (z.B. für eine Person). Ein Feld umfaßt mehrere Symbole (mindestens eines), wobei für jedes Symbol ein byte im Rechner reserviert wird.

Die Summe aller Zeilen (bzw. Spalten) der Datenmatrix nennt man Datei (file).

Bei der sequentiellen Speicherung wird innerhalb der Datei (bei der es sich beispielsweise um einen Datenbestand handeln mag) Datensatz an Datensatz gehängt, also zum Beispiel Kundenadresse an Kundenadresse: Eine lange Adresse nimmt viel Platz in Anspruch, eine kurze hingegen wenig.

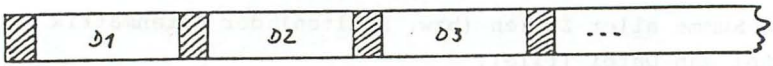
Schematisch sieht dies folgendermaßen aus (D1, D2, D3, ... sind die verschiedenen Datensätze, also z.B. Adressen):



Bei der sequentiellen Speicherung erfordert das Aufsuchen eines ganz bestimmten Datensatzes (z.B. die Adresse von Herrn Müller-Lüdenscheid) viel Zeit, weil dazu die Datei von ihrem Startpunkt an durchsucht werden muß (sie wird "gespult").

Bei der random access-Speicherung (man spricht auch von "Speicherung mit direktem Zugriff"; random access = Zufallszugriff; ein etwas irreführender Name) hat jeder Datensatz die gleiche Länge.

Schematisch sieht dies also folgendermaßen aus:



Auf diese Weise wird zwar Speicherplatz verschenkt (auch kurze Adressen beanspruchen viel Platz), aber der Rechner kann nun auf bestimmte Datensätze direkt zugreifen, ohne die davor befindlichen erst "wegspulen" zu müssen. Das Auffinden eines bestimmten Datensatzes dauert jetzt (bei der Benutzung eines Diskettenspeichers) nur noch Bruchteile von Sekunden.

2.3.2 Tips bei der Initialisierung

In Kapitel 1 wurden schon die wesentlichen Diskettenanweisungen besprochen. Eine besondere Rolle spielte dabei das Kommando

```
INIT
```

Es dient dazu, neue Disketten zu formatieren, um sie im weiteren dann benutzen zu können.
Diese Initialisierung kann man so gestalten, daß bei jedem Laufwerkstart mit dieser Diskette eine Bildschirmausgabe erfolgt, die über die eingelegte Diskette wissenswerte Informationen mitteilt. Dazu müssen in das Initialisierungsprogramm entsprechende PRINT-Anweisungen aufgenommen werden.

2.3.3 Interaktive Diskettennutzung

Einer der wesentlichen Vorteile der Computernutzung in der Datenverarbeitung besteht in der interaktiven Verwendung der externen Speicher.

Wir verstehen darunter den Zugriff auf Dateien der Diskette vom laufenden Programm aus; sei es, daß sich ein Programm benötigte Inputinformationen von der Diskette holt; sei es, daß berechnete Ergebnisse aus dem laufenden Programm heraus zur Archivierung oder zur weiteren, späteren Verwendung auf die Diskette geschrieben werden.

Für diese Art der Diskettennutzung benötigt man zwei zusätzliche BASIC-Anweisungen, nämlich OPEN und CLOSE.

Diese beiden Anweisungen dienen dazu, den Informationsaustausch zwischen Rechner und externem Speicher zu regulieren, indem mit der OPEN-Anweisung der "Kanal" zur Diskette geöffnet und mit der CLOSE-Anweisung wieder geschlossen wird. Genau genommen geht es bei OPEN und bei CLOSE um das Öffnen und Schließen einer Datei (file), also eines Bereichs auf der Diskette, der einen zusammengehörigen Datenbestand umfaßt, und der vom Benutzer mit einem Namen versehen werden muß.

Ein solcher Dateiname, der im Gegensatz zu vielen anderen Rechnern nicht in Anführungszeichen eingeschlossen wird, kann bis zu dreißig Zeichen lang sein.

Das folgende schematische Beispiel zeigt, wie eine solche Datei beschrieben werden kann (wir greifen dabei auf

das Beispiel der Datenmatrix zurück, das weiter oben vorgestellt wurde):

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 6)"2.2 - SCHREIBEN AUF DISKETTE"
60  PRINT : PRINT : PRINT : PRINT
80  PRINT "DIESES PROGRAMM ZEIGT, WIE EIN DATENBE-"
90  PRINT "STAND (DATA IN 500 FF.) INTERAKTIV AUF"
100  PRINT "EINE DISKETTE GESCHRIEBEN WERDEN KANN."
120  GOSUB 1000: REM  WARTEN
130  REM  EINLESEN
140 N = 4
150  DIM NR(N),G$(N),JG(N),F$(N)
160  FOR I = 1 TO N: READ NR(I),G$(I),JG(I),F$(I): NEXT I
170  HOME : PRINT "KONTROLLE : ": PRINT : PRINT
180  FOR I = 1 TO N: PRINT NR(I); TAB( 5)G$(I); TAB( 15)JG(I);
TAB( 22)F$(I): NEXT I
190  PRINT : PRINT : PRINT "ES WIRD JETZT AUF DIE DISKETTE ":
PRINT "GESCHRIEBEN."
200 D$ =  CHR$ (4): REM  CTRL UND D
210  PRINT D$;"OPEN DATEI1"
220  PRINT D$;"WRITE DATEI1"
230  FOR I = 1 TO N: PRINT NR(I),G$(I),JG(I),F$(I): NEXT I
240  PRINT D$;"CLOSE DATEI1"
250  PRINT : PRINT : PRINT "ENDE DER DATENUEBERTRAGUNG"
260  END
500  DATA  001,MAENNLICH,1942,VERHEIRATET
510  DATA  002,WEIBLICH,1953,LEDIG
520  DATA  003,MAENNLICH,1962,LEDIG
530  DATA  004,MAENNLICH,1958,VERHEIRATET
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!"
1020  NORMAL
1030  GET A$: IF A$ = "" THEN 1030
1040  RETURN
```

In diesem Programm geschieht folgendes:

Sätze	Inhalt
3Ø- 6Ø	Überschrift.
8Ø-1ØØ	Erläuternder Text.
12Ø	Sprung ins UP 1ØØØ (Warteprogramm).
14Ø-16Ø	Einlesen der Daten (Angabe der Zahl der Datensätze + Leseschleife über je vier Variablen).
17Ø-18Ø	Kontrollausdruck der Daten.
19Ø	Ankündigung der Übertragung auf die Diskette.
2ØØ	Einführung der Hilfsgröße D$ (der Druck der Tasten CTRL und D gemeinsam entspricht der ASCII-Codezahl 4; deshalb wird der Einfachheit halber CHR$(4) benutzt).
21Ø	In einem String, der der Anweisung PRINT D$ (s. 2ØØ) folgt, wird die Datei, in der die Daten eingefügt werden sollen, geöffnet.
22Ø	In entsprechender Weise wird mitgeteilt, daß die Datei beschrieben (und nicht etwa gelesen) werden soll.
23Ø	Schreibschleife (alle PRINT-Anweisungen gehen jetzt an die Diskette).
24Ø	Schließen der Datei (vergl. 21Ø)
25Ø-26Ø	Endausdruck und Ende.
5ØØ-53Ø	Daten
1ØØØ	Überschrift Warteprogramm.
1Ø1Ø	Mitteilung an den Benutzer in inverser Schrift in Zeile 23 ab Spalte 6.
1Ø2Ø	Zurück von inverser zu normaler Schrift.
1Ø3Ø	Abwarten eines Tastendrucks.
1Ø4Ø	Rücksprung.

Entsprechend läßt sich auch ein einfaches Programm zum Lesen einer Diskettendatei schreiben:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 6)"2.3 - LESEN VON DER DISKETTE"
60  PRINT : PRINT : PRINT : PRINT
70  DIM N$(4)
80  PRINT "DIESES PROGRAMM ZEIGT, WIE EIN DATENBE-"
90  PRINT "STAND VON EINER DISKETTE GELESEN WERDEN": PRINT
TAB( 17)"KANN."
100  REM  SCHREIBEN
110  DATA  OTTO,SUSI,BERTA,EMIL
120  FOR I = 1 TO 4: READ N$(I): NEXT I
130 D$ =  CHR$ (4)
140  PRINT D$;"OPEN DATEI2"
150  PRINT D$;"WRITE DATEI2"
160  FOR I = 1 TO 4: PRINT N$(I): NEXT I
170  PRINT D$;"CLOSE DATEI2"
190  REM  LESEN VON DER DISKETTE
210  PRINT D$;"OPEN DATEI2"
220  PRINT D$;"READ DATEI2"
230  FOR I = 1 TO 4: INPUT A$:N$(I) = A$: NEXT I
240  PRINT D$;"CLOSE DATEI2"
250  HOME : PRINT "KONTROLLE : ": PRINT : PRINT
260  FOR I = 1 TO 4
270  PRINT N$(I)
280  NEXT I
290  PRINT : PRINT : PRINT "ENDE DER AUSGABE": END
```

Programmbeschreibung

Sätze	Inhalt
3Ø- 6Ø	Überschrift.
7Ø	Dimensionierung.
8Ø- 9Ø	Erläuterung.
1ØØ-17Ø	Erzeugen einer Textdatei (zu den Einzelheiten vergl. Programm 2.2 des vorangegangenen Beispiels).
19Ø-24Ø	Lesen von der Diskette (notwendig dazu sind INPUT-Anweisungen; zu den Einzelheiten vergl. vorangegangenes Beispiel).
25Ø-28Ø	Kontrollausgabe.
29Ø	Ende des Programms.

Anmerkung:

Im interaktiven Diskettenbetrieb führt die GET-Anweisung häufig zu Fehlern.
Deshalb haben wir auf ein Unterprogramm zum Abwarten, wie es im vorangegangenen Beispiel benutzt wurde, verzichtet.

Wichtig ist also, daß beim Beschreiben von Disketten im interaktiven Betrieb die Anweisung WRITE, beim Lesen die Anweisung INPUT benutzt wird, die beide als Stringbestandteile in PRINT-Anweisungen auftauchen.
Der aufmerksame Leser wird sicherlich bemerkt haben, daß diese beiden Beispiele sich auf die Benutzung sequentieller Dateien bezogen.
Will man den Informationsaustausch zwischen Rechner und Diskette auf dem Bildschirm sichtbar machen, was für Kontrollzwecke manchmal sinnvoll ist, so gelingt dies mit der Anweisung

```
MON C,I,O
```

Das Argument C in diesem Kommando bewirkt, daß alle internen Betriebssystemanweisungen im Augenblick ihrer Ausführung auf dem Bildschirm erscheinen, I zeigt alle Informationen auf dem Wege von der Diskette zum Rechner (Input), O zeigt alle Informationen auf dem Wege vom Rechner zur Diskette (Output). Einzelne dieser Argumente können auch weggelassen werden.

Die Unterdrückung dieser Ausgaben gelingt mit der Anweisung

```
NOMON
```

oder durch Neustart der Diskette.
Das Kommando NOMON kann die gleichen Argumente enthalten wie das Kommando MON.
Enthält es hingegen weniger Argumente, so wird nur der jeweilige Effekt "abgeschaltet".

Wenn der Leser bei der Erprobung dieser Programme eine Diskette benutzt, auf die er in anderen Zusammenhängen schon Programme (per SAVE-Kommando) gespeichert hat, wird er bei der Ausgabe des Disketteninhaltsverzeichnisses per CATALOG feststellen, daß die "normalen" BASIC-Programme mit einem A gekennzeichnet sind. Diejenigen Dateien hingegen, die im interaktiven Betrieb durch ein laufendes Programm auf der Diskette kreiert und etabliert worden sind, werden im CATALOG durch den Buchstaben T gekennzeichnet.
Man nennt solche Dateien "Textdateien". Solche Dateien können nicht durch LOAD geladen werden, sondern nur im Rahmen von laufenden Programmen, wie es im letzten Beispiel gezeigt wurde.

Etablierte Textdateien können leicht verändert werden, wobei es insbesondere um folgende Aktivitäten geht:

1. Löschen von Dateien;
2. Verlängerung von Dateien;
3. Löschen einzelner Datensätze in der Datei;
4. Veränderung einzelner Datensätze.

Die für diese Aktivitäten notwendigen Anweisungen sollen im folgenden anhand kurzer Beispiele vorgestellt werden.

Das Löschen von Dateien - auch von interaktiv erstellten Textdateien - erfolgt einfach mit dem Kommando

DELETE Dateiname

Wie eine Dateiverlängerung erfolgen kann, zeigt das folgende Programm:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 8)"2.4 - DATEIVERLAENGERUNG"
60  PRINT : PRINT : PRINT : PRINT
70  DIM N$(4),B$(6)
80  PRINT "DIESES PROGRAMM ZEIGT, WIE EIN DATENBE-"
90  PRINT "STAND AUF EINER DISKETTE VERLAENGERT"
100  PRINT "WERDEN KANN."
105  REM  SCHREIBEN
110  DATA  OTTO,SUSI,BERTA,EMIL
120  FOR I = 1 TO 4: READ N$(I): NEXT I
130 D$ =  CHR$ (4)
140  PRINT D$;"OPEN DATEI3"
150  PRINT D$;"WRITE DATEI3"
160  FOR I = 1 TO 4: PRINT N$(I): NEXT I
180  PRINT D$;"CLOSE DATEI3"
190  REM  LESEN VON DER DISKETTE
210  PRINT D$;"OPEN DATEI3"
220  PRINT D$;"READ DATEI3"
230  FOR I = 1 TO 4
240  INPUT A$:B$(I) = A$
250  NEXT I
260  PRINT D$;"CLOSE DATEI3"
270  REM  KONRTROLLAUSDRUCK
280  HOME : PRINT "KONTROLLE : ": PRINT : PRINT
290  FOR I = 1 TO 4
300  PRINT B$(I)
310  NEXT I
320  REM  VERLAENGERUNG
330  PRINT D$;"APPEND DATEI3"
340  PRINT D$;"WRITE DATEI3"
350  PRINT "ADAM": PRINT "EVA"
360  PRINT D$;"CLOSE DATEI3"
370  PRINT : PRINT : PRINT "KONTROLLE NACH VERLAENGERUNG :":
PRINT
380  PRINT D$;"OPEN DATEI3"
390  PRINT D$;"READ DATEI3"
400  FOR I = 1 TO 6: INPUT A$:B$(I) = A$: NEXT I
420  PRINT D$;"CLOSE DATEI3"
430  FOR I = 1 TO 6: PRINT B$(I): NEXT I
440  END
```

Im einzelnen geschieht hier folgendes:

Sätze	Inhalt
3Ø-1ØØ	Überschrift, Erläuterungen und Dimensionierung.
1Ø5-12Ø	Bereitstellen der Daten.
13Ø-18Ø	Etablierung der Diskettendatei und Einschreiben der Daten.
19Ø-26Ø	Lesen der Daten von der Diskette kontrollhalber.
27Ø-31Ø	Kontrollausgabe auf dem Bildschirm.
32Ø-36Ø	Verlängerung der Diskettendatei.
37Ø-42Ø	Einlesen von der Diskette.
43Ø	Kontrollausdruck der verlängerten Datei auf dem Bildschirm.
44Ø	Ende des Programms.

Wir haben uns bei dieser Programmbeschreibung etwas knapper gefaßt, weil die Details der Dateibehandlung ja schon in früheren Beispielen erläutert wurden.

Entscheidend ist hier die Anweisung

APPEND Dateiname

die sowohl die Datei öffnet (also die OPEN-Anweisung ersetzt) als auch für das Anfügen zusätzlicher Informationen sorgt.

Will man nun einzelne Datensätze behandeln (etwa löschen oder ändern), so ist dies nur bei random-access-Dateien möglich. Dazu das folgende Programmbeispiel:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 8)"2.5 - DATEIVERAENDERUNG"
60  PRINT : PRINT : PRINT : PRINT
70  DIM N$(4),B$(4)
80  PRINT "DIESES PROGRAMM ZEIGT, WIE EIN DATENBE-"
90  PRINT "STAND AUF EINER DISKETTE VERAENDERT"
100  PRINT "WERDEN KANN."
105  PRINT : PRINT "EINGEGEBEN WERDEN DIE NAMEN :": PRINT :
PRINT "OTTO, SUSI, BERTA, EMIL"
107  PRINT : PRINT "BERTA SOLL IN EVA VERAENDERT WERDEN."
108  PRINT : PRINT  TAB( 14)"MOMENT BITTE": FOR I = 1 TO 8000:
NEXT I
110  DATA  OTTO,SUSI,BERTA,EMIL
120  FOR I = 1 TO 4: READ N$(I): NEXT I
130 D$ =  CHR$ (4)
140  PRINT D$;"OPEN DATEI4,L10"
150  FOR I = 1 TO 4: PRINT D$;"WRITE DATEI4,R";I
160  PRINT N$(I): NEXT I
180  PRINT D$;"CLOSE DATEI4"
190  REM  LESEN VON DER DISKETTE
210  PRINT D$;"OPEN DATEI4,L10"
220  FOR I = 1 TO 4: PRINT D$;"READ DATEI4,R";I
240  INPUT A$:B$(I) = A$
250  NEXT I
260  PRINT D$;"CLOSE DATEI4"
270  REM  KONRTROLLAUSDRUCK
280  HOME : PRINT "KONTROLLE : ": PRINT : PRINT
290  FOR I = 1 TO 4
300  PRINT B$(I)
310  NEXT I
320  REM  VERAENDERUNG
330  PRINT D$;"OPEN DATEI4,L10"
340  PRINT D$;"WRITE DATEI4,R";3
350  PRINT "EVA"
360  PRINT D$;"CLOSE DATEI4"
370  REM  KONTROLLE
380  PRINT D$;"OPEN DATEI4,L10"
390  FOR I = 1 TO 4
400  PRINT D$;"READ DATEI4,R";I
410  INPUT A$:B$(I) = A$: NEXT I
420  PRINT D$;"CLOSE DATEI4"
430  PRINT : PRINT : PRINT "KONTROLLE NACH VERAENDERUNG :":
PRINT
440  FOR I = 1 TO 4: PRINT B$(I): NEXT I
450  PRINT : PRINT "ENDE": END
```

Auch hier können wir uns bei der Programmbeschreibung kurz fassen:

Sätze	Inhalt
3Ø-1Ø7	Überschrift, Dimensionierung und Erläuterungen.
1Ø8	Warteschleife.
11Ø-12Ø	Dateneingabe.
13Ø-18Ø	Speichern.
19Ø-31Ø	Lesen und Kontrollausdruck.
32Ø-36Ø	Veränderung des Dateiinhalts.
37Ø-44Ø	Lesen und Kontrollausdruck.
45Ø	Ende.

Sicherlich kann sich der Leser jetzt auch vorstellen, wie einzelne Datensätze gelöscht werden können.

2.3.4 Beispiel: Statistische Auswertung

Wie schon an anderer Stelle ausgeführt wurde, entfalten Mikrocomputersysteme ihre Nützlichkeit insbesondere bei der Verwaltung, Bearbeitung und Analyse größerer Datenbestände.

Die Arbeiten, die die Statistiker bis vor wenigen Jahren nur per Hand oder allenfalls unter Benutzung der teuren (und meist überlasteten) Großrechner erledigen konnten, sind jetzt rasch und zuverlässig durch die Kleinrechner zu bewältigen.

Allerdings setzen solche Datenauswertungen voraus, daß der zu analysierende Datenbestand in geschickter und möglichst bequemer Weise, die zudem noch zuverlässig Fehler verhindern soll, dem Rechner zur Verfügung gestellt wird.

Am Beispiel etwa von Umfrageergebnissen läßt sich, wie schon in vorangegangenen Abschnitten, diese Aufgabenstellung anschaulich illustrieren.

Ein derartiges Beispiel eignet sich ganz besonders für dieses Kapitel, weil die Analyse größerer Datenbestände in der Tat nur mit dem Einsatz peripherer Speicher möglich ist. Es werden also z.B. Disketten notwendigerweise eingesetzt werden müssen, weil die auszuwertenden Daten ja auf Dauer oder zumindest für längere Zeit zur Verfügung stehen müssen.

Am Beispiel einer einfachen statistischen Auswertung sollen die notwendigen Arbeitsschritte nun erörtert werden.

Aufgabenstellung

Bei einer Umfrage sind eine Reihe numerischer Daten erhoben worden, nämlich:

1. Alter (in Jahren)
2. Größe (in cm)
3. Gewicht (in kg)
4. Oberer Blutdruckwert
5. Zigarettenkonsum (im Tagesdurchschnitt)

Die einzelnen Aufgaben, die zu erledigen sind, sind die folgenden:

1. Eingabe der Daten auf Diskette
2. Eventuelle Datenkorrektur
3. Kontrollausgabe aller Datensätze
4. Alternative Ausgabe von Sätzen oder Arrays auf Wunsch des Benutzers
5. Bestimmung der Mittelwerte und Standardabweichung für einzelne Variablen (nach Wunsch des Benutzers)
6. Bestimmung von Korrelationskoeffizienten

Statistische Methoden

Zu den Methoden der statistischen Auswertung ist folgendes anzumerken:

Ein Mittelwert (arithmetisches Mittel) für eine Variable berechnet sich folgendermaßen

$$\bar{X} = \frac{1}{n} \sum_{i} X_i$$

wobei $\bar{X}$ = arithmetisches Mittel

X_i = Merkmalswert Nr. i der Variablen X

i = Laufindex

n = Anzahl der Personen (Merkmalsträger)

Eine Standardabweichung als Maß der Streuung der Merkmalswerte einer Variablen berechnet sich wie folgt:

$$s = \sqrt{\frac{1}{n} \sum (X_i - \bar{X})^2}$$

wobei s = Standardabweichung
(übrige Symbole s.o.)

Ein <u>Korrelationskoeffizient</u> bemißt die Stärke des Zusammenhangs zwischen zwei Variablen X und Y:

$$r = \frac{n \sum_i X_i Y_i - \sum_i X_i \sum_i Y_i}{\sqrt{\left[n \sum_i X_i^2 - \left(\sum_i X_i\right)^2\right]\left[n \sum_i Y_i^2 - \left(\sum_i Y_i\right)^2\right]}}$$

wobei r = Korrelationskoeffizient nach Bravais/Pearson

$(-1 <= r <= +1)$

übrige Symbole s.o. (für zwei Variablen X und Y)

Daten

Zur Illustration der weiteren Vorgehensweise benutzen wir die folgenden "Spieldaten":

Person	Alter	Größe	Gewicht	Blutdruck	Zigaretten
1	2o	182	8o	142	3o
2	3o	174	72	134	Ø
3	25	178	8o	14o	2o
4	45	168	68	132	1o
5	42	177	75	13o	Ø
6	58	169	7o	145	2o
7	37	176	82	15o	25

Natürlich hat man es in der Praxis in der Regel mit einer größeren Datenmatrix zu tun, aber das ändert ja nichts an der grundsätzlichen Problematik.

Auf den folgenden Seiten stellen wir das Programm vor, das die gestellten Aufgaben bewältigt.

In diesem Programm fällt ins Auge, daß jeder einzelne eingegebene Datensatz sofort gespeichert wird, nachdem dem Programmbenutzer die Gelegenheit geboten wurde, eventuelle Eingabefehler zu korrigieren.
Dieses direkte Speichern hat einen tieferen Sinn: Würde man nämlich alle Datensätze eingeben wollen, um sie erst dann zu speichern, wäre das Risiko von Datenverlusten (man denke z.B. an einen Stromausfall) unvergleichlich höher.

Es ist zu beachten, daß der erste Datensatz per OPEN-Anweisung, die späteren per APPEND-Anweisung an die Diskette übergeben werden müssen.

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 5)"2.7 - STATISTISCHE AUSWERTUNG"
60  PRINT : PRINT
70  PRINT : PRINT : PRINT "DIESES PROGRAMM DIENT DER ERFASSUNG"
80  PRINT "STATISTISCHER DATEN UND IHRER ERSTEN"
90  PRINT  TAB( 7)"STATISTISCHEN AUSWERTUNG."
95  PRINT : PRINT : PRINT "IN DIESEM PROGRAMM WERDEN NUR NUMERI-":
PRINT "SCHE VARIABLEN BERUECKSICHTIGT."
97  PRINT : PRINT "MISSING VALUES WERDEN NICHT BERUECK-": PRINT
"SICHTIGT."
100  GOSUB 1000: REM  WARTEN
110  INPUT "ANZAHL DER PERSONEN      : ";N
120  PRINT : INPUT "ANZAHL DER VARIABLEN     : ";V
130  DIM W(N,V),X(N),Y(N)
140  FOR I = 1 TO N
150  PRINT : PRINT : PRINT
160  PRINT "PERSON NR. ";I: PRINT : PRINT
170  FOR J = 1 TO V
180  PRINT  TAB( 5)"VAR.NR. ";J;: INPUT " : ";W(I,J)
190  NEXT J
200  PRINT : PRINT : PRINT "SIND DIE ANGABEN KORREKT ";: INPUT
"(J/N) ";A$
210  IF A$ = "J" THEN 230
220  GOSUB 2000: REM  KORREKTUR
230  IF I = 1 THEN  GOSUB 3000: REM  SPEICHER1
240  GOSUB 4000: REM  SPEICHER2
250  NEXT I
260  HOME : PRINT "KONTROLLAUSGABE": PRINT : PRINT
270  PRINT "PERSON"; TAB( 8)"A"; TAB( 13)"GR"; TAB( 18)"GW";
TAB( 23)"BLUT"; TAB( 28)"ZIG": PRINT
280  FOR I = 1 TO N
290  PRINT  TAB( 3)I; TAB( 7)W(I,1); TAB( 12)W(I,2); TAB( 18)W(I,3);
TAB( 23)W(I,4); TAB( 28)W(I,5)
300  IF I / 10 =  INT (I / 10) THEN  GOSUB 1000: REM  WARTEN
310  NEXT I
320  GOSUB 1000: REM  WARTEN
330  PRINT "SOLL EIN BESTIMMTER SATZ AUSGEGEBEN": INPUT "WERDEN
? (J/N) ";A$
340  IF A$ = "N" THEN 400
```

```
350  PRINT : PRINT "WELCHER SATZ ? (BITTE NUMMER) ";: INPUT
I
360  GOSUB 5000: REM  SATZ
370  GOTO 330
400  HOME : PRINT "SOLL EIN BESTIMMTER ARRAY AUSGEGEBEN": INPUT
"WERDEN ? (J/N) ";A$
410  IF A$ = "N" THEN 500
420  PRINT : PRINT "WELCHER ARRAY ? (BITTE NUMMER) ";: INPUT
J
430  GOSUB 6000: REM  ARRAY
440  GOTO 400
500  REM  STATISTISCHE BERECHNUNGEN
505  PRINT : PRINT : PRINT : PRINT
510  PRINT "STATISTISCHE BERECHNUNGEN (J/N) ";: INPUT A$
520  IF A$ = "N" THEN 900
530  PRINT : PRINT : PRINT "MITTELWERTE ? (J/N) ";: INPUT A$
540  IF A$ = "N" THEN 600
550  PRINT : PRINT : INPUT "WELCHE VARIABLE (BITTE NUMMER) ";J
560  FOR I = 1 TO N:X(I) = W(I,J): NEXT I: GOSUB 7000: REM
SCHNITT
570  GOSUB 1000: REM  WARTEN
580  GOTO 530
600  HOME : PRINT "STANDARDABWEICHUNGEN ? (J/N) ";: INPUT A$
610  IF A$ = "N" THEN 700
620  PRINT : PRINT : PRINT "WELCHE VARIABLE (BITTE NUMMER) ";:
INPUT J
630  FOR I = 1 TO N:X(I) = W(I,J): NEXT I
640  GOSUB 7000: REM  SCHNITT
650  GOSUB 8000: REM  SIGMA
660  GOTO 600
700  HOME : PRINT "KORRELATIONEN ? (J/N) ";: INPUT A$
710  IF A$ = "N" THEN 900
720  PRINT : PRINT : PRINT "BITTE VARIABLEN ANGEBEN : ": PRINT
: PRINT
730  INPUT "ERSTE VARIABLE   : ";J1
740  PRINT : INPUT "ZWEITE VARIABLE  : ";J2
750  FOR I = 1 TO N:X(I) = W(I,J1):Y(I) = W(I,J2): NEXT I
760  GOSUB 9000: REM  KORR
770  GOTO 700
900  PRINT : PRINT : PRINT "ENDE DES PROGRAMMS": END
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!": NORMAL
1020  GET A$: IF A$ = "" THEN 1020
1030  HOME : RETURN
```

```
2000  REM  UP KORREKTUR
2010  PRINT : PRINT "WELCHE ANGABE IST FALSCH ?"
2020  INPUT "BITTE NUMMER ANGEBEN : ";J
2030  PRINT : PRINT  TAB( 5)"FALSCHER WERT : ";W(I,J)
2040  PRINT : PRINT  TAB( 5)"KORREKTER WERT: ";: INPUT W(I,J)
2050  HOME : PRINT "KONTROLLE": PRINT : PRINT : PRINT "PERSON
NR. ";I: PRINT : PRINT
2060  FOR J = 1 TO V: PRINT "VAR.NR. ";J; TAB( 15)W(I,J): NEXT
J
2070  PRINT : PRINT "ALLES KORREKT ? (J/N) ";: INPUT A$
2080  IF A$ = "N" THEN  PRINT : GOTO 2010
2090  RETURN
3000  REM  UP SPEICHER1
3010 D$ =  CHR$ (4)
3020  PRINT D$;"OPEN DATEI5"
3030  PRINT D$;"WRITE DATEI5"
3040  FOR J = 1 TO V: PRINT W(I,J): NEXT J
3050  PRINT D$;"CLOSE DATEI5"
3060  RETURN
4000  REM  UP SPEICHER2
4010 D$ =  CHR$ (4)
4020  PRINT D$;"APPEND DATEI5"
4030  PRINT D$;"WRITE DATEI5"
4040  FOR J = 1 TO V: PRINT W(I,J): NEXT J
4050  PRINT D$;"CLOSE DATEI5"
4060  RETURN
5000  REM  UP SATZ
5010  HOME : PRINT "PERSON NR. ";I: PRINT : PRINT
5020  FOR J = 1 TO V: PRINT "VAR.NR. ";J; TAB( 15)W(I,J): NEXT
J
5030  GOSUB 1000: REM  WARTEN
5040  RETURN
6000  REM  UP ARRAY
6010  HOME : PRINT "ARRAY NR. ";J: PRINT : PRINT
6020  FOR I = 1 TO N: PRINT "PERSON NR. ";I; TAB( 15)W(I,J):
NEXT I
6030  GOSUB 1000: REM  WARTEM
6040  RETURN
7000  REM  UP SCHNITT
7010 S = 0
7020  FOR I = 1 TO N:S = S + X(I): NEXT I
7030 AM = S / N
7040  PRINT : PRINT : PRINT "MITTELWERT DER VAR.NR. ";J;" :
"; INT (AM * 100 + .5) / 100
7060  RETURN
```

```
8000  REM  UP SIGMA
8010 S = 0
8020  FOR I = 1 TO N:D = X(I) - AM:D = D * D:S = S + D: NEXT
I
8030 SA =  SQR (S / N):S =  INT (S * 100 + .5) / 100
8040  PRINT : PRINT : PRINT "STANDARDABWEICHUNG        : ";SA
8050  GOSUB 1000: REM  WARTEN
8060  RETURN
9000  REM  UP KORR
9010 S1 = 0:S2 = 0:S3 = 0:S4 = 0:S5 = 0
9020  FOR I = 1 TO N:S1 = S1 + X(I) * Y(I):S2 = S2 + X(I):S3
= S3 + Y(I):S4 = S4 + X(I) * X(I):S5 = S5 + Y(I) * Y(I): NEXT
I
9030 Z = N * S1 - S2 * S3
9040 NN =  SQR ((N * S4 - S2 * S2) * (N * S5 - S3 * S3))
9050 R = Z / NN:R =  INT (R * 1000 + .5) / 1000
9060  PRINT : PRINT : PRINT "KORRELATION ZWISCHEN"
9070  PRINT : PRINT "VAR.NR. ";J1;" UND"
9080  PRINT : PRINT "VAR.NR. ";J2;"  = ";R
9090  GOSUB 1000: RETURN
```

Bei der Programmbeschreibung beschränken wir uns auf die Aufzählung der Hauptbestandteile des Programms und überlassen es dem Leser, durch Lesen des Programms (dies ist übrigens eine sehr wichtige Übung mit hervorragendem Trainingseffekt bei der Verbesserung der BASIC-Kenntnisse) die Einzelheiten zu studieren.

Programmbeschreibung

Sätze	Inhalt
3Ø-97	Überschrift und Erläuterungen.
1ØØ-25Ø	Eingabe der Daten, Fehlerkorrektur und Speichern auf Diskette.
26Ø-32Ø	Kontrollausgabe.
33Ø-37Ø	Ausgabe eines Datensatzes.
4ØØ-44Ø	Ausgabe eines Arrays.
5ØØ-58Ø	Mittelwertberechnungen.
6ØØ-66Ø	Berechnung von Standardabweichungen.
7ØØ-77Ø	Berechnung von Korrelationskoeffizienten.
9ØØ	Programmende.
1ØØØ-1Ø3Ø	UP Warten.
2ØØØ-2Ø9Ø	UP Korrektur falscher Eingaben.
3ØØØ-3Ø6Ø	UP Speichern per OPEN.
4ØØØ-4Ø6Ø	UP Speichern per APPEND.
5ØØØ-5Ø4Ø	UP Ausgabe eines Datensatzes.
6ØØØ-6Ø4Ø	UP Ausgabe eines Arrays.
7ØØØ-7Ø6Ø	UP Mittelwertberechnung.
8ØØØ-8Ø6Ø	UP Standardabweichung.
9ØØØ-9Ø9Ø	UP Korrelationskoeffizient.

Damit der Leser schließlich beurteilen kann, was dieses Programm leistet, seien für die oben vorgestellten "Spieldaten" die Ergebnisse vorgestellt, die nach der erfolgreichen Dateneingabe produziert werden können.

Ergebnisse

Nach der Dateneingabe produziert das Programm zunächst einen Kontrollausdruck des gesamten Datenbestands. Dann können alle Mittelwerte, alle Standardabweichungen und alle bivariaten Korrelationskoeffizienten bestimmt werden. Dabei ergibt sich:

		Mittelwert	Standard-abweichung
1	Alter	36.71	12.o2
2	Größe	174.86	4.61
3	Gewicht	75.29	5.o9
4	Blutdruck	139	6.78
5	Zigaretten	15	11.o2

Korrelationskoeffizienten

		Alter 1	Größe 2	Gewicht 3	Blutdr. 4	Zigar. 5
1	Alter	1	-o.825	-o.664	-o.oo5	-o.237
2	Größe		1	o.848	o.16	o.337
3	Gewicht			1	-o.542	-o.573
4	Blutdruck				1	o.831
5	Zigaretten					1

Es ist leicht einsichtig, daß ein solches schon recht umfangreiches Programm unter mehreren Gesichtspunkten verbessert und verschönert werden könnte. Einige wenige dieser Aspekte sollen hier genannt werden.

1. Wenn die Daten - wie hier vorgesehen - auf Diskette gespeichert werden, dann könnte natürlich vorgesehen werden, daß sie bei der statistischen Auswertung nicht alle im Arbeitsspeicher gehalten werden (bei großen Datenbeständen geht das auch gar nicht), sondern daß die jeweils zu bearbeitenden Arrays nach den entsprechenden Angaben des Benutzers erst von der Diskette eingelesen werden.

   Durch Einbau entsprechender Leseprogramme ist dies leicht zu realisieren.

2. Für den praktischen Einsatz müßte vorgesehen werden, daß bei den Berechnungen eventuelle missing values (Antwortverweigerer u.ä.) übergangen werden. Mit simplen IF-Abfragen und ggf. Verringerung der "echten" Zahl der Antworten kann dies erreicht werden.

3. Nicht immer hat man es ausschließlich mit numerischen Variablen zu tun:

   Treten auch Stringvariablen auf, so könnte nicht mit W(N,V) dimensioniert werden, sondern die einzelnen Variablen müßten dann als eindimensionale Arrays angelegt werden.

   Die Schleifen, die von J=1 bis V laufen (z.B. beim Eingeben der Daten, aber auch an einigen anderen Stellen), müßten dann in einzelne Statements "aufgelöst" werden.

4. Schließlich würde es die Ergebnisse lesbarer machen, wenn z.B. die Variablen nicht einfach nur numeriert würden, sondern wenn die inhaltlichen Bezeichnungen benutzt würden ("Alter", "Größe" usw.)

   Dazu müßten diese Begriffe z.B. über DATA-Statements in einen Begriffs-Array BØ(J) (J=1 bis V) eingelesen und dann an passender Stelle immer mitbenutzt werden.

Diese Anmerkungen sollen verdeutlichen, daß bei echten Datenverarbeitungsprogrammen doch ein beträchtlicher Aufwand betrieben werden muß (auch wenn dies nicht sehr schwierig ist), wenn man zu komfortablen Programmen gelangen will.

Kapitel 3: Speicherzugriff und Maschinenprogramme

3.1 Direkter Speicherzugriff

Die in den ersten beiden Kapiteln besprochenen BASIC-Sprachelemente reichen in den meisten Fällen für die anliegenden Programmierprobleme aus. Für einen etwas fortgeschritteneren Benutzer kommt es dann aber nicht mehr nur darauf an, irgendeine Lösung seines Problems zu finden, sondern es spielen auch andere Aspekte eine Rolle: Die Eleganz der Programmierung, die Ausführungsgeschwindigkeit und die Übersichtlichkeit der Ergebnisse sollen hier als Beispiele genannt werden. Sobald jemand vom "Computerfieber" gepackt ist, sucht er nach der optimalen Lösung. Da es unterschiedliche Programmierstile gibt, wie es unterschiedliche Sprachstile gibt, ist Optimalität in diesem Zusammenhang nicht objektiv zu definieren. Dies ist ein Grund, weshalb wir die Optimalität nicht in den Vordergrund dieses Buches gestellt haben. Doch so verschieden die Vorstellungen von einem "guten" Programm auch sein mögen, man wird bemüht sein, alle Möglichkeiten seines Rechners auszunutzen, um zu einem besseren Programm zu gelangen, ganz gleich was man darunter versteht.

Bislang haben wir in allen Beispielen dem Rechner die Betreuung seines Speichers überlassen.

Der Speicherinhalt wurde nur über BASIC-Befehle wie LIST oder PRINT abgefragt. Der APPLE II bietet aber auch die Möglichkeit, den Inhalt einzelner Speicherstellen direkt abzufragen oder sogar abzuändern. Um dies sinnvoll tun zu können, muß man wissen, was ein bestimmter Wert in einer bestimmten Speicherstelle bedeutet.
Es sollen im folgenden einige Möglichkeiten vorgestellt werden, wie man den direkten Speicherzugriff sinnvoll in Programmen einsetzen kann. Wiederum ist beabsichtigt, dem Leser Anregungen zu geben, nicht jedoch eine vollständige Liste von Einsatzmöglichkeiten zu erstellen.

Bei den benötigten Sprachelementen handelt es sich um den Befehl

```
POKE  adresse,wert
```

mit dem man an die angegebene Adresse den angegebenen Wert schicken kann, sowie um die Funktion

```
PEEK (adresse)
```

durch die der Inhalt der angegebenen Speicherstelle abgefragt wird. In beiden Fällen muß die Adresse dezimal angegeben werden. Auch der Wert im POKE-Befehl ist eine Dezimalzahl.

Beispiele:

```
POKE   33,30
POKE   32,5
PRINT PEEK(222)
PRINT PEEK(1Ø3)+PEEK(1Ø4)*256
```

Durch die Befehlsfolge der beiden POKE-Anweisungen wird zunächst die Bildschirmbreite auf 3Ø reduziert, dann wird der linke Bildschirmrand auf Spalte 5 verschoben, d.h. mit Hilfe der Speicherstellen 32 und 33 kann die Breite eines Bildschirmfensters festgelegt. werden.
Der linke und rechte Teil des Schirms sind so "gesperrt", sie können nicht mehr beschrieben werden. Auch der HOME-Befehl wirkt nur noch innerhalb des Fensters.

PEEK muß wie jede Funktion mit dem PRINT-Befehl gekoppelt sein, wenn ihr Funktionswert sichtbar werden soll. Wie man sieht, kann sie auch in Rechenformeln verwendet werden. Die Speicherstelle 222 enthält den Code des Fehlers der zuletzt aufgetreten ist.
Im obigen Beispiel wird also eine Dezimalzahl auf dem Bildschirm angezeigt. Das letzte Beispiel berechnet aus 2 Teilen einer Hexadezimalzahl die entsprechende Dezimalzahl, und zwar enthalten die Speicherstellen 1Ø3 und 1Ø4 diejenige Speicheradresse, an der das aktuelle Programm beginnt, dies ist meist 2Ø49.

Anwendungen

1. Wenn ein Programm neu eingegeben wird, so beginnt die Speicherung in der Regel an der Stelle 2Ø49 (hexadezimal $Ø8Ø1). Der unterhalb dieser Adresse liegende RAM-Bereich (bis $Ø8ØØ) dient dem Betriebssystem. Diese Adresse "merkt sich" der Rechner in folgender Weise: $Ø8 wird umgewandelt in die dezimale Zahl 8, $Ø1 in die dezimal Zahl 1.
   Die höherwertigen 2 Ziffern der hexadezimalen Adresse speichert er ab an der Stelle 1Ø4, die niederwertigen an der Stelle 1Ø3. So konnte man durch die schon genannte Formel die Zahl 2Ø49 für den Beginn des Programms feststellen.
   Die nächste freie Speicherstelle nach dem Ende des Programms im Speicher wird - ähnlich aufbereitet - den Speicherstellen 176 und 175 zugewiesen, d.h. der Befehl

```
PRINT  PEEK(175)+PEEK(176)*256
```

   bringt die letzte Adresse, die das Programm benötigt, plus 2, auf den Bildschirm.

   Die Kenntnis dieser Adressen kann man ausnutzen, um sein Programm zu "verstecken".
   Dazu sei folgendes Vorgehen angeregt: Der Leser überzeuge sich durch LIST, daß sich ein Programm im Arbeitsspeicher befindet (ansonsten sollte eines geladen oder eingetippt werden).

Danach gebe er folgende Befehle im Direktmodus ein:

```
POKE 1Ø3,PEEK(175):POKE 1Ø4,PEEK(176)

POKE PEEK(1Ø3)+256*PEEK(1Ø4)-1,Ø
```

Wenn nun LIST eingegeben wird, ist das vorher sichtbare Programm verschwunden. Der Bildschirm ist entweder leer oder es erscheinen Reste eines vorher anscheinend durch NEW gelöschten Programms. Das vorher eingegebene Programm kann wieder sichtbar gemacht werden durch:

```
POKE 1Ø3,1:POKE 1Ø4,8
```

Im Kapitel 7 finden sich Beispiele, wie diese "Spielerei" mit großem praktischen Nutzen eingesetzt wird von den Programmen des APPLE-DOC.
Dort erfolgen solche Neubelegungen von Speicherstellen, auch der obengenannten.

2. Schon bei der Vorstellung der Befehle wurde das <u>Bildschirmfenster</u> erwähnt. Durch Ansprechen und Neubelegung der entsprechenden Speicherstellen kann der benutzbare Bildschirm verkleinert werden. Dies ist insbesondere dann interessant, wenn Teile des Bildschirms gegen Überschreiben oder "Wegrollen" gesichert werden sollen. Zuständig dafür sind die Speicherstellen 32 bis 35.

32: Legt den linken Bildschirmrand fest. die Werte der Stelle 32 müssen zwischen Ø und 39 liegen.

33: Gibt die Bildschirmbreite an, nicht den rechten Rand. Zulässige Werte liegen zwischen 1 und 4Ø, es muß darauf geachtet werden, daß PEEK(33) + PEEK(32) nicht größer als 4Ø werden darf. Es ist sinnvoll, zunächst 33 und dann erst 32 zu verändern.

34: Oberer Bildschirmrand in Werten zwischen Ø und 23. Angegeben werden muß die erste freie Zeile.

35: Unterer Bildschirmrand in Werten zwischen Ø und 23. Angegeben wird die erste gesperrte Zeile.

Die Einsatzmöglichkeiten sind vielfältig. So hätten wir beispielsweise den "Kopf" aller unserer Beispielprogramme so programmieren können:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 40: PRINT "-";: NEXT : POKE 34,5: HOME
```

Für eventuell folgende Bildschirmanzeigen stehen nur noch 18 Zeilen zur Verfügung, da durch POKE 34,6 die Zeilen Ø bis 5 auf dem Bildschirm gesperrt sind. Auch der HOME-Befehl löscht nur innerhalb des Bildschirmfensters.

Etwas ähnliches kann man nun auch am unteren Rand machen. Wenn in der letzten Zeile der Name des gerade arbeitenden Programms während des gesamten Durchlaufs sichtbar bleiben soll, könnte man das Beispielprogramm etwa so erweitern:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 40: PRINT "-";: NEXT : POKE 34,6: HOME

10  VTAB 22: FOR I = 1 TO 40: PRINT "-";: NEXT : PRINT
: HTAB 9: PRINT "PROGRAMM: DEMO TEXTFENSTER";: POKE 35,21:
HOME
```

Die letzte verfügbare Zeile ist nun die Zeile 2Ø, d.h. es stehen uns nur noch 15 Zeilen zur Verfügung.
Die Begrenzungen des Bildschirms werden erst wirksam nach dem ersten Befehl, der eine neu gesperrte Speicherstelle anspricht, deshalb wurden die HOME-Befehle am Ende der Zeilen 5 und 1Ø aufgenommen. Auf diese Weise ist gesichert, daß nachfolgende PRINT-Statements nur innerhalb des Fensters positioniert werden.

Nun wird das Programm erweitert um einige Ausdruckbefehle und INPUT-Statements. Es werden 5 Eingaben verlangt für die Variablen A1 bis A5, deren Durchschnitt dann berechnet wird. Die Anzeigen sollen jedoch etwas mehr in die Mitte des Schirms gerückt werden, was im nachfolgenden Programm mit TAB(6) erreicht wurde:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 40: PRINT "-";: NEXT : POKE 34,6: HOME

10  VTAB 22: FOR I = 1 TO 40: PRINT "-";: NEXT : PRINT
: HTAB 9: PRINT "PROGRAMM: DEMO TEXTFENSTER";: POKE 35,21:
HOME
20  PRINT  TAB( 6)"1. EINGABE  : ";: INPUT "";A1
30  PRINT  TAB( 6)"2. EINGABE  : ";: INPUT "";A2
40  PRINT  TAB( 6)"3. EINGABE  : ";: INPUT "";A3
50  PRINT  TAB( 6)"4. EINGABE  : ";: INPUT "";A4
60  PRINT  TAB( 6)"5. EINGABE  : ";: INPUT "";A5
70 X = (A1 + A2 + A3 + A4 + A5) / 5
80  PRINT : PRINT : PRINT  TAB( 6)"DURCHSCHNITT: ";X
90  END
```

Wenn sehr viele solcher verschobenen Bildschirmanzeigen erfolgen sollen, ist es bequemer, statt die TAB-Funktion in jeder Zeile zu Anfang des Ausdruckteils zu benutzen, den linken Rand des Bildschirms zu versetzen:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 40: PRINT "-";: NEXT : POKE 34,6: HOME

10  VTAB 22: FOR I = 1 TO 40: PRINT "-";: NEXT : PRINT
: HTAB 9: PRINT "PROGRAMM: DEMO TEXTFENSTER";: POKE 35,21:
HOME
20  POKE 33,35: POKE 32,5: HOME
30  INPUT "1. EINGABE  : ";A1
40  INPUT "2. EINGABE  : ";A2
50  INPUT "3. EINGABE  : ";A3
60  INPUT "4. EINGABE  : ";A4
70  INPUT "5. EINGABE  : ";A5
80 X = (A1 + A2 + A3 + A4 + A5) / 5
90  PRINT : PRINT : PRINT "DURCHSCHNITT: ";X
100  END
```

Es besteht im Ergebnis kein Unterschied zur vorhergehenden Version, aber der Programmierer hat Tipparbeit gespart, das Programm ist etwas kürzer geworden und bei der Ausführung entfallen 6 TAB-Befehle, d.h. es wird Zeit eingespart.
Dies macht sich natürlich erst in größeren Programmen mit vielen solchen Programmzeilen spürbar bemerkbar.

Eine andere Möglichkeit für den Einsatz von POKE 32,X besteht in dem Fall, daß erst der linke Teil des Bildschirms beschrieben werden soll und dann der rechte.
So könnten etwa die Eingaben für 6 Variablen auch nebeneinander gesetzt werden, erst 3 links, dann 3 rechts. Dies geschieht in der folgenden Version:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 40: PRINT "-";: NEXT : POKE 34,6: HOME

10  VTAB 22: FOR I = 1 TO 40: PRINT "-";: NEXT : PRINT
: HTAB 9: PRINT "PROGRAMM: DEMO TEXTFENSTER";: POKE 35,21:
HOME
20  INPUT "1.EINGABE: ";A1
30  INPUT "2.EINGABE: ";A2
40  INPUT "3.EINGABE: ";A3
50  POKE 33,20: POKE 32,20: HOME
60  INPUT "4.EINGABE: ";A4
70  INPUT "5.EINGABE: ";A5
80  INPUT "6.EINGABE: ";A6
90  PRINT "SUMME: ";A1 + A2 + A3 + A4 + A5 + A6
100  END
```

Die Zeilen 2Ø bis 4Ø arbeiten wie gewohnt, am linken Rand des Schirms beginnt der Text.
Durch Zeile 5Ø wird die Bildschirmbreite auf 2Ø reduziert und der linke Rand auf Position 2Ø gesetzt. Wichtig ist wieder die Reihenfolge:

Erst wird die Breite reduziert, dann wird der Rand versetzt. Der HOME-Befehl hat wieder die schon genannte Aufgabe, die vorhergehenden POKE-Anweisungen wirksam werden zu lassen.
Dadurch werden die Anzeigen, die durch die Zeilen 6Ø bis 8Ø bewirkt werden, neben die vorherigen (aus den Zeilen 2Ø bis 4Ø) placiert. Auch der Ergebnisausdruck aus Zeile 9Ø steht im rechten Teil des Bildschirms ab Spalte Nr. 2Ø (d.h. aber der 21. Spalte).

Sperrungen von Bildschirmbereichen können natürlich auch rückgängig gemacht werden, indem man POKE-Befehle mit den ursprünglichen Werten der Speicherstellen verwendet. Ein letztes Beispiel soll dies veranschaulichen anhand der vorherigen Programmversion; es bietet sich an, den Bildschirm vor dem Ergebnisausdruck wieder zu verbreitern, da die Summe ja aus allen 6 Eingaben berechnet wurde:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 40: PRINT "-";: NEXT : POKE 34,6: HOME

10  VTAB 22: FOR I = 1 TO 40: PRINT "-";: NEXT : PRINT
: HTAB 9: PRINT "PROGRAMM: DEMO TEXTFENSTER";: POKE 35,21:
HOME
20  INPUT "1.EINGABE: ";A1
30  INPUT "2.EINGABE: ";A2
40  INPUT "3.EINGABE: ";A3
50  POKE 33,20: POKE 32,20: HOME
60  INPUT "4.EINGABE: ";A4
70  INPUT "5.EINGABE: ";A5
80  INPUT "6.EINGABE: ";A6
90  PRINT : PRINT
100  POKE 32,0: POKE 33,40
110  PRINT : PRINT "DIE SUMME BETRAEGT: ";A1 + A2 + A3
+ A4 + A5 + A6
120  END
```

Nach den schon bekannten Funktionen der Zeile 3 bis 8Ø werden zunächst 2 Leerzeilen eingeschoben (allerdings nur im rechten Bildschirmbereich). Danach werden der Bildschirmrand und die Bildschirmbreite wieder auf die normalen Werte Ø und 4Ø gesetzt. Dabei muß nun genau die umgekehrte Reihenfolge eingehalten werden wie vorher.
Der erste PRINT-Befehl in Zeile 11Ø wirkt zunächst noch nur auf den rechten Teil des Schirms; deshalb mußte er dem Ergebnisausdruck vorangesetzt werden, damit dieser nun wieder am linken Bildschirmrand beginnt.

3. Eine dritte Anwendungsmöglichkeit betrifft ebenfalls den Bildschirm, aber nicht das Bildschirmfenster. Der Leser sollte einmal folgendes Programm starten:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HOME : FOR J = 1 TO 24: FOR I = 1 TO 40: PRINT "A";:
NEXT I,J
20  FOR Q = 1 TO 2000: NEXT
30 X = 160
40  FOR Q = 1024 TO 2039: POKE Q,X: NEXT
50  END
```

Das Programm füllt zunächst den gesamten Bildschirm mit dem Buchstaben "A" als Folge von Zeile 1Ø. Zeile 2Ø enthält eine "leere" Schleife der Länge 2ØØØ, die nur dafür sorgt, daß der gefüllte Bildschirm etwas länger betrachtet werden kann. Durch die Schleife in Zeile 4Ø wird nun der Bildschirm wieder gelöscht, allerdings nicht zeilenweise von oben nach unten, sondern erst die Zeile Ø, dann Zeile 8, Zeile 16, dann Zeile 1, Zeile 9, Zeile 17, dann Zeile 2, Zeile 1Ø, Zeile 18 usw. Durch POKE Q,X wird der Bildschirmzeichencode für das Leerzeichen (16Ø) in alle Speicherstellen von 1Ø24 bis 2Ø39 geschrieben, dies ist der Speicherbereich für den Bildschirminhalt. In 1Ø24 bis 1Ø63 stehen die Codes für alle Zeichen der ersten Zeile, in 1Ø64 bis 11Ø3 die der neunten Zeile, in 11Ø4 bis 1143 die der siebzehnten Zeile.
Danach werden 8 Stellen "übersprungen" und die Stellen 1152 bis 1191 enthalten die Codes aller Zeichen der zweiten Zeile usw.

Der Bildschirmbereich ist also eingeteilt in 3 Gruppen: Zeile Ø bis 7, Zeile 8 bis 15 und Zeile 16 bis 23. Die Abspeicherung des Inhalts erfolgt dann so: eine Zeile der ersten Gruppe, eine der zweiten, eine der dritten, danach 8 Stellen für andere Informationen, dann wieder eine Zeile der ersten Gruppe, eine der zweiten, eine der dritten, 8 Stellen "frei" usw., d.h. die Speicherstellen für das jeweils erste Zeichen von 2 aufeinanderfolgenden Zeilen einer Gruppe haben Adressen, die sich gerade

um 128 unterscheiden. Dies wird im nächsten Programm ausgenutzt.

Im ersten Beispiel wurde der direkte Speicherzugriff eingesetzt, um einen "vollen" Bildschirm zu löschen. Man kann aber natürlich auch durch den POKE-Befehl ein Zeichen an eine ganz bestimmte Stelle des Bildschirms setzen, bzw. eine Folge von Zeichen an einer bestimmten Stelle beginnen lassen. Das wird am folgenden Programm demonstriert, das den gleichen Bildschirminhalt erzeugt wie die Zeilen 2Ø bis 11Ø des vorletzten Programms.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 2500: NEXT
10  HOME : PRINT :A$ = ".EINGABE: ":L =  LEN (A$)
20 A = 1664:E = 3:I = 1
30  FOR QQ = 0 TO 1
40 Q = A:F$ =  STR$ (I)
50  POKE Q, ASC (F$) + 128
60  FOR Z = 1 TO L:Q = Q + 1:F$ =  MID$ (A$,Z,1): POKE
Q, ASC (F$) + 128: NEXT
70  POKE Q + 1,96
80  POKE 37,0: PRINT
90  INPUT "";X:S = S + X
100 B$ =  STR$ (X):LX =  LEN (B$)
110  FOR Z = 1 TO LX:Q = Q + 1:F$ =  MID$ (B$,Z,1): POKE
Q, ASC (F$) + 128: NEXT
120  POKE 37,0
130 A = A + 128:I = I + 1: IF I <  = E THEN 40
140 A = 1684:E = 6
150  NEXT QQ
160 A$ = "DIE SUMME BETRAEGT: ":L =  LEN (A$)
170 Q = 1447
180  FOR Z = 1 TO L:Q = Q + 1:F$ =  MID$ (A$,Z,1): POKE
Q, ASC (F$) + 128: NEXT
190 A$ =  STR$ (S):L =  LEN (A$)
200  FOR Z = 1 TO L:Q = Q + 1:F$ =  MID$ (A$,Z,1): POKE
Q, ASC (F$) + 128: NEXT
210  FOR I = 1024 TO 1036: POKE I,160: NEXT
220  END
```

Programmbeschreibung

Sätze	Inhalt
1Ø-2Ø	Es werden die Anfangswerte einiger Variablen festgelegt. A$ erhält den immer gleichbleibenden Teil des Ausgabestrings zugewiesen und seine Länge wird in L gespeichert. Die Variable A steht für Adresse, ihr Anfangswert beträgt 1664; dies ist die Speicherstelle für das erste Zeichen der 6. Bildschirmzeile (Zeile Nr. 5). Dort soll die Anzeige beginnen. E steht für die letzte Positionsnummer einer Eingabe im linken bzw. rechten Teil des Schirms, d.h. E ist zunächst auf 3 zu setzen. I ist der Zähler für die Eingaben, beginnt also mit dem Wert 1.
3Ø	Beginn einer Schleife mit 2 Durchläufen für die Unterscheidung in 2 Bildschirmteile.
4Ø	Q ist die aktuell anzusprechende Speicherstelle, sie wird mit dem Anfangswert der Adressen A belegt. F$ ist der String, der die Ziffer für die Eingabenummer I enthält.
5Ø	Der Code für das Bildschirmzeichen aus dem String F$ (hier Eingabenummer) wird an die Stelle Q geschrieben (hier Anfangsadresse A).
6Ø	Durch die Anweisungsfolge werden die Codes für alle Zeichen des Strings A$ (hier A$ = ".EINGABE: ") in aufeinanderfolgende Speicherstellen geschrieben (deshalb wird Q jeweils um 1 erhöht). Da es sich um Zeichen in einer Zeile handeln soll, liegen die zugehörigen Adressen unmittelbar hintereinander.

Sätze	Inhalt
7Ø	Hinter dem Text soll, wie als Folge des erweiterten INPUT-Befehls, ein blinkendes, invers ausgegebenes Leerzeichen erscheinen. Dies hat den Code 96, der also in die nächstfolgende Speicherstelle geschrieben wird.
8Ø	Damit der Cursor durch den INPUT-Befehl der Zeile 9Ø nicht auf dem Bildschirm nach unten wandert, wird er durch POKE 37,Ø an die erste Stelle oben links positioniert, das nachfolgende PRINT dient sozusagen nur der Initialisierung des POKE-Befehls.
9Ø	Die Eingabe eines Wertes erfolgt nun in der oberen linken Ecke des Schirms; außerdem wird summiert.
1ØØ-11Ø	Ab der Stelle des blinkenden Leerzeichens werden nun alle Ziffern der eingegebenen Zahl für X(gegebenenfalls auch der Dezimalpunkt) in den Bildschirmspeicher geschrieben. Die Befehlsfolge ist im wesentlichen schon aus der Zeile 6Ø bekannt. Statt A$ muß jedoch die Variable B$ für den aus X erzeugten String verwendet werden, da der Wert von A$ im weiteren Verlauf des Programms noch benötigt wird.
12Ø	Der Cursor wird wieder nach oben gesetzt.
13Ø	Nun muß zur nächsten Zeile gewechselt werden, deshalb wird A verändert. Da die nächste Zeile zur gleichen Gruppe von Bildschirmzeile gehört, kann dies durch die Addition von 128 erfolgen, wie schon vorher erläutert wurde.

Sätze	Inhalt
	Zudem wird der Zähler für die Eingaben um 1 erhöht und es wird geprüft, ob die letzte Eingabe in diesem Bildschirmteil schon erfolgt ist. Wenn nicht, so werden die Zeilen 4Ø bis 13Ø erneut durchlaufen mit verändertem I und verändertem Anfangswert für Q; diese letzte Festsetzung erfolgt in Zeile 4Ø.
14Ø	Falls I einen Wert größer als E erreicht hat, sind die Eingaben auf einer Bildschirmseite gemacht worden und A muß nun wieder zurückgesetzt werden auf die Adresse 1684, dies ist die Adresse für das 21. Zeichen der 6. Zeile, und E, die Obergrenze für die Positionsnummern der Eingaben, muß auf 6 erhöht werden. I enthält bereits den neuen Wert 4.
15Ø	Sämtliche Anweisungen werden jetzt noch einmal wiederholt, und zwar für die Beschriftung des rechten Bildschirmteils, also für die vierte bis sechste Eingabe.
16Ø	Die Stringvariable A$ erhält einen neuen Wert, nämlich den Text des Ergebnisausdrucks und die Variable L für seine Länge wird entsprechend neu belegt.
17Ø	Q wird auf 1447 gesetzt, dies ist die letzte Stelle vor dem ersten Zeichen der 12. Zeile. (Das entspricht dann den 3 PRINT-Statements aus dem vorletzten Programm).
18Ø	Der Text aus A$ wird zeichenweise in die hintereinanderliegenden Speicherstellen ab 1448 geschrieben (da Q als erstes um 1 erhöht wird). Die so erzeugte Bildschirmanzeige beginnt also beim ersten Zeichen der 12. Zeile. Durch Verändern der Zeile 17Ø kann auch ein etwas eingerückter Ausdruck erreicht werden, z.B. durch die Festlegung Q=1453.

Sätze	Inhalt
19Ø-2ØØ	Die berechnete Summe S wird ziffernweise in die anschließenden Speicherstellen geschrieben.
21Ø	Da die erste Zeile des Schirms nach all diesen Operationen noch Ziffern von der letzten Eingabe (Zeile 9Ø für I=6) enthält,werden die ersten 13 Bildschirmstellen durch Leerzeichen ersetzt. Denkbar wäre auch, zur Sicherheit die gesamte Zeile neu mit Leerzeichen zu beschreiben. Dazu müßte der Endwert der Schleife von 1Ø36 auf 1Ø63 erhöht werden.
22Ø	Ende des Programms

Es sollen noch einige Anweisungen des vorstehenden Programms etwas näher erläutert werden.
Die Zeilen 8Ø und 12Ø enthalten einen POKE-Befehl für die Speicherstelle 37. Der Wert dieser Stelle gibt die aktuelle vertikale Cursorposition an, also die Bildschirmzeile, in der der Cursor sich gerade befindet. Zulässig sind Werte zwischen Ø und 23.

Eine kurze Bemerkung noch zu den Funktionen STR$ und ASC.
STR$ benötigt als Argument einen numerischen Ausdruck, d.h. eine Zahl, eine numerische Variable oder eine Rechenformel. Durch die Funktion wird die Zahl in einen String verwandelt, d.h. sie wird vom Rechner als Zeichenfolge aus Ziffern angesehen. Dadurch können die Stringbearbeitungsfunktionen LEFT$, RIGHT$ und MID$ eingesetzt werden.

Die Funktion wird im obigen Programm benötigt, da ASC nur auf Strings angewendet werden kann. Denn diese Funktion wandelt das erste Zeichen eines als Argument angegebenen Strings um in die zugehörige ASCII-Codezahl.
Da sie nur ein Zeichen jeweils "übersetzt", mußte in unserem Programm mehrfach auf Schleifen und die Funktion MID$(A$,Z,1) zurückgegriffen werden.

Es ist darauf hinzuweisen, daß der Begriff "ASCII-Codezahl" nicht eindeutig ist. Darin liegt auch der Grund dafür, daß in den Zeilen 50,60,110,180 und 200 nicht ASC(F$) gepokt wird, sondern dieser Wert noch um 128 erhöht wird. Dies liegt daran, daß die Code-Zahlen der Tastatur, über die ja die Zeichen eingegeben werden, andere sind als die Code-Zahlen des Bildschirms.

Dazu ein Beispiel: Der Befehl

```
PRINT ASC(" ")
```

hat die Ausgabe der ASCII-Codezahl für das Leerzeichen zur Folge, die Anzeige ist 32.
Durch POKE 1192,32 wird dieser Wert in die Speicherstelle für das 1. Zeichen der 1o. Bildschirmzeile geschrieben. Dort wird aber nicht das übliche Leerzeichen sichtbar, sondern ein invers geschriebenes Leerzeichen. Der Bildschirm muß drei Typen des Leerzeichens auseinanderhalten: ein normales, ein inverses und ein blinkendes, die Tastatur hingegen nicht.
Dies gilt für alle Zeichen.

Üblicherweise übernimmt der Rechner die Zuordnung. Ist weder der Befehl INVERSE noch FLASH gegeben worden, muß zu allen Codes, die von der Tastatur kamen, 128 addiert werden. Ist hingegen INVERSE wirksam, kann der Code unmittelbar übernommen werden. Ist FLASH inkraft, wird der Rechner 64 addieren. Der aktuelle Ausgabemodus ist übrigens in der Speicherstelle 5Ø zu überprüfen:
Inhalt 63 entspricht INVERSE,
Inhalt 127 entspricht FLASH,
Inhalt 255 entspricht NORMAL.

Bei direktem Speicherzugriff wird die Übersetzungshilfe des Rechners umgangen, deshalb muß der Benutzer diese Aufgabe übernehmen. In den Handbüchern des APPLE finden sich die entsprechenden Code-Tabellen.

Das letzte Programm hat wohl die Ausnutzung des direkten Speicherzugriffs für die Beschriftung des Bildschirms deutlich gemacht, aber andererseits doch auch gezeigt, daß dadurch die Programmierung in manchen Fällen recht mühsam werden kann. Dies wäre noch deutlicher geworden, wenn die angesprochenen Bildschirmzeilen nicht zu einer Gruppe gehört hätten, wie im Beispiel zur Vereinfachung angenommen wurde. Jedoch verschafft einem ein solches Programm einen besseren Einblick in die Arbeitsweise des Rechners, zum zweiten eröffnen solche Kenntnisse oft erst die Möglichkeit, bestimmte Programmieraufgaben zu bewältigen.

Als sehr gutes Anwendungsbeispiel dafür kann das Programm MASKENGENERATOR aus dem Abschnitt 7.4 angesehen werden. Dort sind die Kenntnisse über den Bildschirmspeicherbereich überaus nützlich eingesetzt worden; ohne diese Kenntnisse wäre die Programmstruktur nicht denkbar.

4. Eine vierte Anwendungsmöglichkeit des direkten Speicherzugriffs betrifft die Tastatur.
Besonders häufig wird die Kenntnis über die Tastatur-Speicherstellen benötigt bei folgender Aufgabenstellung:
Ein Programm soll an einer bestimmten Stelle angehalten werden, um dem Benutzer Gelegenheit zu geben, eine Bildschirmanzeige in Ruhe zu betrachten. So könnte etwa das Programm Informationen zu seiner Bedienung ausgeben, die der Benutzer vor dem Weiterarbeiten erst durchlesen muß. Oder es folgen mehrere Ergebnistabellen hintereinander; dann ist eine Unterbrechung nötig, damit die Ergebnisse überhaupt wahrgenommen werden können, insbesondere wenn ein Drucker nicht eingesetzt werden kann. Oder das Programm liefert Fehlermeldungen, die der Benutzer vor der Fortsetzung des Programmablaufs zur Kenntnis nehmen muß. Eine andere Situation ist der Fall, daß das Programm durch einen Tastendruck abgebrochen wird.
Für beide Fälle sollen kleine Beispielprogramme vorgestellt werden.

Soll ein Programm angehalten werden, bis der Benutzer eine Fortsetzung wünscht, kann natürlich der Befehl

```
STOP
```

an geeigneter Stelle im Programm eingefügt werden. Nach der Eingabe von

```
CONT
```

über die Tastatur, wird der Ablauf des Programms genau an der Unterbrechungsstelle wieder aufgenommen:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 2000: NEXT
10  HOME : PRINT "DIESES PROGRAMM DIENT ZUR DEMONSTRATION":
PRINT : PRINT "ES SOLL DEUTLICH MACHEN, AUF WELCHE ART":
PRINT : PRINT "EIN PROGRAMM UNTERBROCHEN UND WIEDER": PRINT
: PRINT "FORTGESETZT WERDEN KANN."
20  PRINT : PRINT : PRINT : PRINT : PRINT "SIE KOENNEN
DAS PROGRAMM FORTSETZEN ": PRINT : PRINT "DURCH EINGABE
VON ";: INVERSE : PRINT " CONT ";: NORMAL : PRINT " UEBER
DIE": PRINT : PRINT "TASTATUR."
30  STOP
40  HOME : PRINT " I"," I*I": PRINT
50  FOR I = 1 TO 50
60  PRINT I,I * I: IF I / 10 <  >  INT (I / 10) THEN  NEXT
I
70  STOP : PRINT : PRINT : IF I <  > 50 THEN  NEXT I
80  PRINT "DAS PROGRAMM IST BEENDET."
90  END
```

Der Benutzer erhält zunächst eine Information über das, was das Programm leistet, danach eine Bedienungshilfe, wie er sich bei Unterbrechung zu verhalten hat. Nach dem ersten STOP in Zeile 3Ø erfolgt eine entsprechende Meldung auf dem Schirm. Nach CONT fährt der Rechner mit der Programmausführung fort. Nun folgt ein einfaches Programm, das die Zahlen von 1 bis 5Ø und ihre Quadratzahlen ausgibt. Nach jeweils 1Ø Zeilen wird das Programm wieder unterbrochen mit einer entsprechenden Meldung und muß mit CONT fortgesetzt werden.

```
DIESES PROGRAMM DIENT ZUR DEMONSTRATION

ES SOLL DEUTLICH MACHEN, AUF WELCHE ART

EIN PROGRAMM UNTERBROCHEN UND WIEDER
FORTGESETZT WERDEN KANN.

SIE KOENNEN DAS PROGRAMM FORTSETZEN

DURCH EINGABE VON  CONT  UEBER DIE
TASTATUR.

BREAK IN 30
]CONT
 I              I*I

1               1
2               4
3               9
4               16
5               25
6               36
7               49
8               64
9               81
10              100

BREAK IN 70
]CONT
DAS PROGRAMM IST BEENDET.
```

Die Meldungen

BREAK IN 3Ø oder BREAK IN 7Ø

verhindern aber eine gute Bildschirmgestaltung; außerdem muß der Benutzer wissen, daß es sich nicht um eine Fehlermeldung handelt. Und nicht zuletzt ist es lästig, immer CONT einzutippen.

Eine erste Verbesserungsmöglichkeit bietet der GET-Befehl, wie er im nachfolgenden Programm eingesetzt wird:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 2000: NEXT
10  HOME : PRINT "DIESES PROGRAMM DIENT ZUR DEMONSTRATION":
PRINT : PRINT "ES SOLL DEUTLICH MACHEN, AUF WELCHE ART":
PRINT : PRINT "EIN PROGRAMM UNTERBROCHEN UND WIEDER": PRINT
: PRINT "FORTGESETZT WERDEN KANN."
30  GOSUB 200
40  HOME : PRINT " I"," I*I": PRINT
50  FOR I = 1 TO 50
60  PRINT I,I * I: IF I / 10 < > INT (I / 10) THEN NEXT
I
70  GOSUB 200: PRINT : PRINT : IF I < > 50 THEN HOME
: PRINT " I"," I*I": PRINT : NEXT I
80  VTAB 22: PRINT "DAS PROGRAMM IST BEENDET."
90  END
200  VTAB 22: HTAB 6: INVERSE : PRINT "DRUECKEN SIE IRGENDEINE
TASTE": NORMAL
210  GET A$: IF A$ = "" THEN 210
220  VTAB 22: CALL - 958: RETURN
```

Programmbeschreibung

Sätze	Inhalt
1Ø	Der Text erläutert, wozu das Programm dient. Die Zeile 2Ø des vorigen Programms ist entfallen. Erklärungen zur Bedienung sind vorab nicht nötig.
3Ø	Sprung ins Unterprogramm ab Zeile 2ØØ, das zur Unterbrechung führt.
4Ø	Die Bildschirmausgabe wird vorbereitet: Der Schirm wird gelöscht und die Überschrift wird angezeigt; eine Leerzeile folgt.
5Ø	Schleifenbeginn
6Ø	Nach Ausgabe des Ergebnisses wird geprüft, ob schon 1Ø Zahlen angezeigt wurden. Wenn nicht, erfolgt die Ausgabe der nächsten Zeile.
7Ø	Sprung ins Unterprogramm, da nach 1Ø Zahlen eine Unterbrechung erfolgen soll. Danach wird geprüft, ob noch ein weiterer Block ausgegeben werden muß. Falls ja, wird der Bildschirm wieder gelöscht und die Überschrift erfolgt mit nachfolgender Leerzeile. Anschließend wird die Schleife fortgesetzt.
8Ø	Falls I in Zeile 7Ø bereits auf dem Höchstwert 5Ø stand, erfolgt in der Bildschirmzeile 21 (in der 22. Zeile) eine Meldung, daß das Programm beendet ist.
9Ø	Ende des Programms

Sätze	Inhalt
2ØØ-22Ø	Unterprogramm zur Unterbrechung
2ØØ	In der 22. Bildschirmzeile erfolgt die Anweisung, wie das Programm fortgesetzt werden kann, nämlich durch einen beliebigen Tastendruck, und zwar erscheint der Text invers und um 5 Stellen nach rechts verrückt. Anschließend wird zum normalen Ausgabemodus zurückgekehrt.
21Ø	Durch den GET-Befehl wird das Programm solange unterbrochen, bis eine Tastatureingabe erfolgt. Solange A$ noch der leere String ist, wird immer wieder die Zeile 21Ø durchlaufen.
22Ø	Nach Löschen der Zeile 22 durch CALL-958 kehrt der Rechner zum Hauptprogramm zurück.

Mit dieser Änderung des Programms hat man zweierlei erreicht: Zum einen reicht nun ein einziger Tastendruck aus, um im Programmablauf fortfahren zu können, zum anderen wird der Bildschirmausdruck nicht durch Meldungen des Rechnerbetriebssystems verdorben.
Außerdem reicht eine Einzeilenmeldung des Programms aus, um den Benutzer über die Bedienung zu informieren.
Trotzdem hat diese Art der Programmierung einen Nachteil. Werden aus Versehen oder aus Unkenntnis zwei Tasten gedrückt, so bleibt das zweite Zeichen im Tastaturpuffer stehen, so daß beim nächsten GET der Rechner nicht mehr auf eine neue Eingabe wartet.

Ein zweiter Nachteil liegt in unerklärlichen Interaktionen begründet zwischen dem GET-Befehl und einigen DOS-Befehlen.
Beiden Problemen kann man bei folgender Programmversion ausweichen:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 2000: NEXT
10  HOME : PRINT "DIESES PROGRAMM DIENT ZUR DEMONSTRATION":
PRINT : PRINT "ES SOLL DEUTLICH MACHEN, AUF WELCHE ART":
PRINT : PRINT "EIN PROGRAMM UNTERBROCHEN UND WIEDER": PRINT
: PRINT "FORTGESETZT WERDEN KANN."
30  GOSUB 200
40  HOME : PRINT " I"," I*I": PRINT
50  FOR I = 1 TO 50
60  PRINT I,I * I: IF I / 10 < > INT (I / 10) THEN NEXT
I
70  GOSUB 200: PRINT : PRINT : IF I < > 50 THEN HOME
: PRINT " I"," I*I": PRINT : NEXT I
80  VTAB 22: PRINT "DAS PROGRAMM IST BEENDET."
90  END
200  VTAB 22: HTAB 6: INVERSE : PRINT "DRUECKEN SIE IRGENDEINE
TASTE": NORMAL
210  POKE - 16368,0: WAIT - 16384,128: POKE - 16368,0
220  VTAB 22: CALL - 958: RETURN
```

Das Hauptprogramm bis Zeile 9Ø ist nicht verändert worden, nur das Unterprogramm und dort nur die Zeile 21Ø.

In der Speicherstelle 49 152 (oder -16 384 als Komplement zu 65 536) wird registriert, ob überhaupt eine Taste gedrückt wurde, bzw. ob der Code für die letzte gedrückte Taste schon weitergeleitet werden konnte. Um vor Überraschungen sicher zu sein, wird der Wert dieser Stelle auf Ø gesetzt, damit nicht alte Tastatureingaben berücksichtigt werden.

Der Befehl WAIT ist eine Spezialität des APPLE, er stellt sozusagen ein bedingtes STOP dar mit nachfolgendem automatischen CONT.
Das Programm bleibt unterbrochen, solange der Inhalt der Speicherstelle nicht mindestens so groß ist wie der im Befehl genannte Wert.
Die allgemeine Form des Befehls ist

```
WAIT adresse,wert
```

Die im Beispiel genannte Adresse -16 384 (bzw. 49 168) enthält Informationen über die letzte Tastatureingabe. Wurde eine Taste gedrückt, steht dort ein Wert, der mindestens 128 ist. Daher die spezielle Form des Befehls.

Wie sieht es nun im zweiten Fall aus, in dem ein Programm solange durchgeführt wird, bis entweder END erreicht ist oder aber eine Unterbrechung durch den Benutzer gewünscht wird?
Wieder wurde das Beispiel des Ausdrucks von Zahlen und ihrer Quadratzahlen gewählt. Allerdings jetzt nicht in Form einer FOR...NEXT-Schleife, sondern als potentielle Endlosschleife:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR Q = 1 TO 2000: NEXT
10  HOME : POKE 37,21: PRINT
20  INVERSE : PRINT "ZUR BEENDIGUNG DES PROGRAMMS:": PRINT
: HTAB 10: PRINT "DRUECKEN SIE IRGENDEINE TASTE";: NORMAL

30  SPEED= 150
40  POKE 35,20: HOME
50  POKE  - 16368,0
60 I = I + 1
70  PRINT I,I * I
80  IF  PEEK ( - 16384)' < 128 THEN 60
90  POKE  - 16368,0
100  SPEED= 255: TEXT
110  END
```

Programmbeschreibung

Sätze	Inhalt
1Ø	Nach dem Löschen des Bildschirms wird der Cursor auf die Zeile 21 positioniert, durch das nachfolgende PRINT wird der Befehl erst wirksam.
2Ø	Dort erfolgt nun eine Bedienungsanleitung für den Benutzer.
3Ø	Durch den Befehl SPEED= kann die Ausgabegeschwindigkeit für den Bildschirm verändert werden. Üblicherweise steht SPEED auf 255. Hier wird eine Reduzierung vorgenommen, damit die nachfolgenden Anzeigen besser lesbar sind.
4Ø	Der untere Teil des Bildschirms wird gesperrt.
5Ø	Genau wie im vorherigen Programm soll vermieden werden, daß alte Tastatureingaben berücksichtigt werden.
6Ø-7Ø	Berechnung und Ausgabe
8Ø	Durch diese bedingte Sprunganweisung wird erreicht, daß die Zeilen 6Ø und 7Ø immer wieder durchlaufen werden, solange wie der Wert der Speicherstelle -16 384 für die Tastatureingabe kleiner als 128 ist. Sobald eine Taste gedrückt wird, steigt der Wert der Speicherstelle auf 128 oder mehr und die "Endlos"-Schleife wird abgebrochen.
9Ø	Da die eben gemachte Eingabe nicht vom Rechner verarbeitet werden soll, wird -16 368 wieder auf Ø gesetzt.

Sätze	Inhalt
1ØØ	Die Ausgabegeschwindigkeit wird wieder auf den üblichen Wert gesetzt. Durch den Befehl TEXT werden POKE-Befehle für das Bildschirmfenster wieder aufgehoben.
11Ø	Ende des Programms

Einsatzmöglichkeiten für Programmunterbrechungen werden noch deutlicher im Kapitel 5, bzw. im Abschnitt 7.4. Doch sollte es auch jetzt schon dem Leser möglich sein, bei einigen seiner Programmieraufgaben die vorstehenden Anregungen aufzugreifen und nützlich einzusetzen.

Bei diesen 4 Bereichen der Anwendung von direktem Speicherzugriff wollen wir es in diesem Kapitel belassen. Es hätten sicher noch eine Vielzahl weiterer Problembereiche aufgeführt werden können, jedoch nur bei Verzicht auf einige Programmbeispiele; dem praktischen Einsatz sollte jedoch der Vorrang vor der Vollständigkeit eingeräumt werden.

3.2. Die Benutzung von ASSEMBLER-Programmen

3.2.1 Grundlagen

Der Benutzer eines Mikrocomputers, wie z.B. des APPLE, geht häufig davon aus, daß die hier im Mittelpunkt stehende Programmiersprache, nämlich die Sprache BASIC, die einzige Möglichkeit darstellt, mit dem APPLE zu kommunizieren.

Dem ist aber nicht so. Es gibt eine ganze Reihe weiterer Programmiersprachen, die teilweise auch beim APPLE verwendet werden können, - zumindest wenn man beispielsweise per Diskette die notwendigen Übersetzungsprogramme (Compiler) bereitstellen kann.

Es ist deshalb ganz sinnvoll, zunächst einmal kurz und in allgemeiner Weise über die verschiedenen Programmiersprachen zu sprechen, um dann speziell auf die Möglichkeiten und den Nutzen von ASSEMBLER einzugehen.

Die ursprüngliche, einem Rechner direkt verständliche Sprache, kann im Grunde nur aus den beiden Symbolen Ø und 1 aufgebaut werden, die stellvertretend für die beiden Zustände "Stromkreis offen" und "Stromkreis geschlossen" stehen. Eine solche Maschinensprache wäre natürlich für uns nur sehr schwer handhabbar.

Daher hat man sich schon sehr früh daran gemacht, solche Sprachelemente zu vereinbaren, die sich der uns geläufigen Symbole bedienen.

Eine dieser Sprachen ist die Sprache ASSEMBLER.

Dabei bedeutet dieser Begriff ASSEMBLER im Grunde zweierlei:

1. Zum einen handelt es sich - wie eben ausgeführt - um eine (maschinenorientierte) Programmiersprache; deshalb haben wir oben Großbuchstaben verwendet.

2. Zum anderen wird auch häufig mit dem gleichen Begriff das Übersetzungsprogramm gemeint, das ein Assembler-Programm in ein Maschinenprogramm "zurückübersetzt".

Die Programmierung in ASSEMBLER macht also ein Übersetzungsprogramm erforderlich, das schlicht auch Assembler heißt und in der Regel per Diskette geladen werden muß.

Die Entwicklung einer solchen, nun etwas benutzerfreundlicheren Computersprache war allerdings auch mit einem Nachteil verbunden:
Die Rückübersetzung vom sog. Quellenprogramm (source program) in das Maschinenprogramm (object program) kostet Rechenzeit und verbraucht Speicherkapazität wegen der Haltung des Übersetzungsprogramms.

Mit zunehmender Leistungsfähigkeit der Rechnung bezüglich der Speicherkapazität und der Rechengeschwindigkeit war dieser Nachteil aber leicht in Kauf zu nehmen.

Im Gegenteil: Man ging noch weiter und entwickelte Programmiersprachen, die noch weitaus benutzerfreundlicher sind als etwa die verschiedenen ASSEMBLER-Sprachversionen.
Diese benutzerorientierten Sprachen zeichneten sich vor allem dadurch aus, daß die umständliche Speicheradressierung, die in ASSEMBLER noch notwendig ist, nun wegfällt. Dadurch sind diese Sprachen, die allerdings wiederum zusätzliche Rückübersetzungsprozesse erforderlich machen, relativ leicht erlernbar und haben sich rasch durchsetzen können.

Die wichtigsten und bekanntesten dieser Sprachen sind die folgenden:

- FORTRAN (Formula Translation):
  Vor allem für wissenschaftlich-technische Probleme geeignet;
- COBOL (Common Business Oriental Language):
  Vor allem im betrieblichen Bereich die generelle Sprache;
- ALGOL (Algorithmic Language):
  Vor allem für mathematische Problemstellungen geeignet.

Zu dieser Gruppe von benutzerorientierten Fragen gehört in neuerer Zeit auch BASIC (Beginner's All-purpose Symbolic Instruction Code).

Die Verwendung dieser Benutzersprachen ist mit wachsenden Rechenzeiten verbunden. Deshalb verlaufen insbesondere solche Prozeduren, die sich aus einer großen Zahl sich wiederholender einfacher Einzelschritte aufbauen, vergleichsweise langsam.

Gerade solche einfachen Arbeitsschritte eignen sich sehr gut dazu, nach wie vor in ASSEMBLER programmiert zu werden, um dann z.B. in BASIC-Programme eingebaut zu werden. Dadurch kann eine Beschleunigung der Gesamtrechenzeiten erreicht werden. Genau damit werden wir uns im folgenden beschäftigen.

In diesem Kapitel geht es nun allerdings nicht darum, aus dem Leser perfekte ASSEMBLER-Programmierer zu machen, sondern es sollen Grundzüge dieser Programmiersprache vermittelt werden, die ein Verständnis ermöglichen für den Einbau von ASSEMBLER-Unterprogrammen in BASIC-Programme.

Um dieses Grundverständnis zu fördern, ist es sinnvoll, zunächst einen kurzen Blick auf zwei in der Datenverarbeitung außerordentlich wichtige Zahlensysteme zu werfen - das Binär- und das Hexadezimalsystem.

Das Binärsystem verfügt nur über die beiden Ziffern Ø und 1. Um nun verstehen zu können, was z.B. 11Ø11Ø1 bedeutet, werfen wir einmal einen Blick auf eine beliebige Dezimalzahl:

2348

Diese Zahl läßt sich zerlegen in

$$2*1ØØØ + 3*1ØØ + 4*1Ø + 8*1$$

$$= 2*1Ø^3 + 3*1Ø^2 + 4*1Ø^1 + 8*1Ø^Ø$$

Daraus können wir folgende generelle Regel entnehmen:

Jede Zahl läßt sich zerlegen in eine Summe, die so viele Summanden wie die zu zerlegende Zahl Ziffern hat. Jeder Summand ist ein Produkt aus zwei Faktoren. Der erste Faktor entspricht der jeweiligen Ziffer der zu zerlegenden Zahl; der zweite Faktor ist eine Zehnerpotenz, wobei die Exponenten ganze Zahlen von Z-1 abwärts darstellen, wenn Z die Ziffernanzahl ist.

Der gleiche Satz gilt auch im Binärsystem (und in jedem anderen Zahlensystem auch), mit dem Unterschied, daß anstelle des Wortes "Zehnerpotenz" nun "Zweierpotenz" stehen muß.

Somit können wir nun auch die Zahl 11Ø11Ø1 erklären (Verwandlung einer Binärzahl in eine Dezimalzahl):

$$11Ø11Ø1 =$$

$$1*2^6 + 1*2^5 + Ø*2^4 + 1*2^3 + 1*2^2 + Ø*2^1 + 1*2^Ø$$

$$= 1*64 + 1*32 + Ø*16 + 1*8 + 1*4 + Ø*2 + 1*1$$

$$= 64 + 32 + 8 + 4 + 1$$

$$= 1Ø9_{(1Ø)}$$

Umgekehrt läßt sich eine beliebige Dezimalzahl in eine Binärzahl umwandeln, indem wir sie mit Zweierpotenzen "auffüllen" - mit den größtmöglichen beginnend.

Was bedeutet 93 im Binärsystem?

$$93_{(1Ø)} =$$

$$\underline{1}*64 + \underline{Ø}*32 + \underline{1}*16 + \underline{1}*8 + \underline{1}*4 + \underline{Ø}*2 + \underline{1}*1$$

$$= 1Ø111Ø1_{(2)}$$

Stellt man einmal die Zahlen des Dezimalsystems und des Binärsystems einander gegenüber, so erhält man

Dezimal	Binär
Ø	Ø
1	1
2	1Ø
3	11
4	1ØØ
5	1Ø1
6	11Ø
7	111
8	1ØØØ
9	1ØØ1

Man erkennt, daß man vierstellige Binärzahlen braucht, wenn die 1Ø Ziffern des Dezimalsystems im Rechner dargestellt werden soll.
Wenn man aber schon 4 Stellen zu reservieren hat, könnte man in diesen Speicherstellen mehr unterbringen als nur 1Ø verschiedene Symbole (Ø, 1, ..., 9), was leicht zu erkennen ist, wenn man in der zweiten Spalte der obigen Tabelle einfach "weiterzählt".

Dezimal	Binär
⋮	⋮
8	1ØØØ
9	1ØØ1
1Ø	1Ø1Ø
11	1Ø11
12	11ØØ
13	11Ø1
14	111Ø
15	1111

Diese Tabelle zeigt, daß auf vier Binärstellen maximal 16 verschiedene Symbole untergebracht werden können (Ø, 1, ..., 15).
Dies erklärt die Bedeutung des Hexadezimalsystems in der Datenverarbeitung. Es garantiert die optimale Ausnutzung der zur Verfügung stehenden Speicher.
Gezählt wird in diesem System mit 16 Ziffern wie folgt:

Dezim.	binär	Hex.	Dezim.	binär	Hex.
Ø	Ø	Ø	11	1Ø11	B
1	1	1	12	11ØØ	C
2	1Ø	2	13	11Ø1	D
3	11	3	14	111Ø	E
4	1ØØ	4	15	1111	F
5	1Ø1	5	16	1ØØØØ	1Ø
6	11Ø	6	17	1ØØØ1	11
7	111	7	18	1ØØ1Ø	12
8	1ØØØ	8	19	1ØØ11	13
9	1ØØ1	9	2Ø	1Ø1ØØ	14
1Ø	1Ø1Ø	A	⋮	⋮	

Die Beziehungen zwischen Dezimal- und Hexadezimalsystem lassen sich am einfachsten wieder an den Umwandlungen von dem einen System ins andere erläutern.

Will man eine Hexadezimalzahl in eine Dezimalzahl umwandeln, so gilt der allgemeine Zerlegungssatz von oben.

Beispiel:

$$1ØFA_{(16)} =$$

$$1*16^3 + Ø*16^2 + F*16^1 + A*16^Ø$$

$$= 1*4Ø96 + Ø + 15*16 + 1Ø*1$$

$$= 4Ø96 + 24Ø + 1Ø = 4346_{(1Ø)}$$

Will man umgekehrt eine Dezimalzahl in eine Hexadezimalzahl verwandeln, so ist sie mit 16er-Potenzen zu füllen.

Beispiel:

$$458_{(10)} =$$

$$1*16^2 + 12*16^1 + 10*16^0$$

$$= \underline{1}*256 + \underline{C}*16 + \underline{A}*1$$

$$= 1CA_{(16)}$$

Die Betrachtung des Hexadezimalsystems ist in diesem Zusammenhang deshalb ganz sinnvoll, weil die ASSEMBLER-Anweisungen, die - wie wir noch sehen werden - aus Worten bestehen, die aus drei Buchstaben aufgebaut werden, in Hexadezimalzahlen verwandelt werden.
Hinzu kommt, daß die Speicheradressen, die - etwa zum Zwecke der internen Datentransformation - angegeben werden müssen, als Hexadezimalzahlen einzugeben sind.

Wir wollen hier die einzelnen ASSEMBLER-Anweisungen nicht besprechen, sondern verweisen dazu auf die entsprechende Spezialliteratur (siehe z.B.: R. MOTTOLA: Assembly Language Programming for the Apple II, Osborne/McGraw Hill, 1982).

Generell gilt, daß jede ASSEMBLER-Anweisung aus fünf Teilen (Feldern) aufgebaut wird (nicht alle fünf Felder müssen notwendigerweise besetzt werden, aber das soll hier nicht interessieren):

ASSEMBLER-Anweisung: Schematischer Aufbau

Satz Nr.	Etikett	Mnemonic	Argument	Kommentar
1	2	3	4	5

Beispiele:

```
1ØØØ    VAL1    EQU     3ØB   ;  ZUORDNUNG(LET)
11ØØ    ADD     LDA     VAL1  ;  Laden des A-
                                     REGISTERS
```

Die zentrale Position ist das Feld 3: Dort steht das Schlüsselwort der ASSEMBLER-Anweisung.
In der Zeile 1ØØØ ist dies die Anweisung EQU, welche bewirkt, daß die Speicheradresse $3ØB_{(16)}$ ($=779_{(1Ø)}$) mit einem Wert belegt wird.

Im ersten Feld steht eine obligatorische Satznummer, an der zweiten Stelle kann (aber muß nicht) ein "Etikett" stehen, welches als "Ansprungetikett" benutzt werden kann, wenn es als Argument (Feld 4) in einer anderen Anweisung benutzt wird (siehe Satz 11ØØ).

In Feld 4 steht ein Argument, das aber nur bei manchen Anweisungen erforderlich ist, - vergleichbar etwa dem INPUT-Statement in BASIC, das ohne Variablennamen als Argument ja nicht verwendet werden könnte - im Gegensatz etwa zur BASIC-Anweisung RETURN.

Im letzten Feld kann ein erläuternder Kommentar angefügt werden. Er ist häufig z.B. durch ; oder durch * abzutrennen - je nachdem, welche ASSEMBLER-Version man benutzt. Bleiben beispielsweise die Felder 1 bis 4 unbesetzt, entstehen reine Kommentaranweisungen, die mit der REM-Anweisung aus BASIC zu vergleichen sind.

Betrachten wir nun einmal ein komplettes ASSEMBLER-Programm, das wir aus dem erwähnten Buch von MOTTOLA entnehmen (S. 25/26).

Das folgende Programm dient dazu, das Wort ERR auf dem Bildschirm auszugeben, veranlaßt zwei akustische Signale über den Lautsprecher des APPLE und verhindert den Zeilenvorschub nach Ausgabe von drei Leerzeichen.

Ein solches Programm könnte sehr gut dazu benutzt werden, um z.B. Eingabefehler anzuzeigen.

Bei diesem Programm wird teilweise auf Maschinen-Unterprogramme zurückgegriffen, über die der APPLE II im ROM-Speicher schon verfügt, nämlich auf Programme, die unter folgenden Startadressen stehen:

$FF2D_{(16)}$: Ausdruck ERR und Piepston

$FBDD_{(16)}$: Piepston

$F948_{(16)}$: Ausgabe von drei Leerzeichen

Für die erste und die dritte Adresse hat der APPLE schon Etiketten zur Verfügung, nämlich PRERR und PRBLNK. Der zweiten Adresse können wir z.B. das Etikett BELL zuordnen (es ist generell empfehlenswert,

schon existierende und dokumentierte Etiketten zu nutzen).

Das Programm könnte nun folgendermaßen aussehen:

```
1  PRBLNK  EQU  F948
2  BELL    EQU  FBDD
3  PRERR   EQU  FF2D
4  ERROR   JSR  PRERR   ;DRUCKE "ERR"
5          JSR  BELL    ;PIEPSEN
6          JSR  PRBLNK  ;3 BLANKS
7          RTS          ;RETURN
8          END
```

In den ersten drei Zeilen erfolgen die notwendigen Zuordnungen zu Etiketten. In den Zeilen 4-6 wird in diejenigen Unterprogramme verzweigt, die die oben genannten Aufgaben erledigen. In Satz 7 erfolgt der Rücksprung und mit Satz 8 wird das Programm beendet.

Dieses ASSEMBLER-Programm tut von sich aus noch nichts - genausowenig wie ein BASIC-Programm etwas tut, bevor das Kommando RUN eingegeben wird.

Bevor ein ASSEMBLER-Programm abgearbeitet werden kann, muß es aber zunächst "assembliert", d.h. in die maschinenverständliche Form zurückübersetzt werden. Dies geschieht nach Eingabe des Kommandos ASM (und natürlich nur dann, wenn dem APPLE über Diskette ein ASSEMBLER-(Übersetzungs-)Programm zur Verfügung gestellt wurde).

Mit dem Kommando ASM wird neben dem Quellenprogramm das Objektprogramm erzeugt, das folgendermaßen aussieht:

```
**END OF PASS 1
**END OF PASS 2

0800            1  ;*************************
0800            2  ;THIS SUBROUTINE WILL
0800            3  ;PRINT "ERR", BEEP
0800            4  ;THE SPEAKER TWICE,
0800            5  ;THEN PRINT 3 BLANKS
0800            6  ;*************************
0800            7  ;
0800            8  ;
0800            9  PRBLNK   EQU  $F948
0800            10 BELL1A   EQU  $FBDD
0800            11 PRERR    EQU  $FF2D
0800            12 ;
0800            13 ;
0800 202DFF     14 ERROR    JSR  PRERR    ;PRINT "ERR"
0803 20DDFB     15          JSR  BELL1A   ;BEEP SPEAKER
0806 2048F9     16          JSR  PRBLNK   ;PRINT 3 BLANKS
0809 60         17          RTS           ;THEN RETURN
                18          END

***** END OF ASSEMBLY

            *************************
            *                       *
            * SYMBOL TABLE -- V 1.5 *
            *                       *
            *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

** ABSOLUTE VARIABLES/LABELS

PRBLNK F948  BELL1A FBDD  PRERR  FF2D  ERROR  0800

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0032
```

Nach der Assemblierung ist erkennbar, daß das Objektprogramm numeriert wird, und zwar mit der Hexadezimalzahl 8ØØ beginnend. Diese Zeilennummer ändert sich nicht, bis zur ersten Zeile des "Programmkerns", d.h. bis zur Zeile mit dem Etikett "ERROR".
Dies bedeutet, daß das eigentliche Objektprogramm nur aus der Zeilengruppe besteht, die mit der ERROR-Zeile beginnt und mit der RETURN-(=RTS-)Zeile endet.
Die übrigen Zeilen sind nicht assembliert worden - und dies ist auch nicht notwendig - genausowenig wie beispielsweise REM-Anweisungen in BASIC-Programmen übersetzt werden.

Es wird in obiger Programmliste deutlich, daß auch die EQU-Zeilen (vergleichbar mit den LET-Anweisungen in BASIC) nicht assembliert worden sind. Gleichwohl werden diese aber berücksichtigt, wie aus der sog. Symboltabelle deutlich wird. EQU erzeugt also keine Objektcode-Anweisung, jedoch wird der ASSEMBLER angewiesen, etwas zu tun. Insoweit nimmt diese Anweisung (wie einige andere auch) quasi eine Zwischenstellung ein zwischen einer Anweisung, die in den Objektcode assembliert wird und einer Anweisung, die völlig übergangen wird (wie z.B. die Kommentar-Anweisungen, die mit ; eingeleitet werden). Man nennt eine derartige Anweisung "Assembler-Direktive" oder "Pseudo-op".

Eine besonders wichtige dieser Direktiven ist die Anweisung ORIGIN, die die Startadresse für den Objektcode festlegt (Voreinstellung für die Startadresse ist beim hier verwendeten LISA-Assembler die Hexadezimaladresse 8ØØ).

Will man eine andere Startadresse verwenden - und dies ist zumindest dann erforderlich, wenn mehrere ASSEMBLER-Programme aus einem Hauptprogramm heraus aufgerufen werden sollen - dann kann dies mit der Anweisung ORG (=ORIGIN) geschehen. Allerdings muß dann auch mitgeteilt werden, ab welcher Startadresse der Code während des Assemblierens (und nicht nur danach) gespeichert werden soll. Nur so kann ein unter Umständen unerfreuliches Durcheinander von Assembler (Übersetzungsprogramm), Quellenprogramm und Objektprogramm vermieden werden.

Die Anweisung, die die Startadresse für das Objektprogramm angibt, heißt OBJ.

Also fügt man z.B. die folgenden beiden Anweisungen am Anfang des Programms ein:

```
ORG 3ØØ ; Neue Startadresse des Programms
OBJ 8ØØ ; Startadresse des Codes während
          der Assemblierung
```

Das Programm sieht jetzt also folgendermaßen aus:

```
1  ;*************************
2  ;THIS SUBROUTINE WILL
3  ;PRINT "ERR", BEEP
4  ;THE SPEAKER TWICE,
5  ;THEN PRINT 3 BLANKS
6  ;*************************
7  ;
8  ;
9          ORG  $300
10         OBJ  $800
11 ;
12 ;
13 PRBLNK  EQU  $F948
14 BELL1A  EQU  $FBDD
15 PRERR   EQU  $FF2D
16 ;
17 ;
18 ERROR   JSR  PRERR      ;PRINT "ERR"
19         JSR  BELL1A     ;BEEP SPEAKER
20         JSR  PRBLNK     ;PRINT 3 BLANKS
21         RTS             ;THEN RETURN
22         END
```

Wenn wir dieses Programm nun assemblieren, so ergibt sich:

```
**END OF PASS 1
**END OF PASS 2

0800          1 ;*************************
0800          2 ;THIS SUBROUTINE WILL
0800          3 ;PRINT "ERR", BEEP
0800          4 ;THE SPEAKER TWICE,
0800          5 ;THEN PRINT 3 BLANKS
0800          6 ;*************************
0800          7 ;
0800          8 ;
0300          9          ORG  $300
0300          10         OBJ  $800
0300          11 ;
0300          12 ;
0300          13 PRBLNK  EQU  $F948
0300          14 BELL1A  EQU  $FBDD
0300          15 PRERR   EQU  $FF2D
0300          16 ;
0300          17 ;
0300 202DFF   18 ERROR   END  PRERR      ;PRINT "ERR"
0303 20DDFB   19         JSR  BELL1A     ;BEEP SPEAKER
0306 2048F9   20         JSR  PRBLNK     ;PRINT 3 BLANKS
0309 60       21         RTS             ;THEN RETURN
              22         END

***** END OF ASSEMBLY

              *************************
              *                       *
              * SYMBOL TABLE -- V 1.5 *
              *                       *
              *************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

** ABSOLUTE VARIABLES/LABELS

PRBLNK F948  BELL1A FBDD  PRERR  FF2D  ERROR  0300

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:0032
```

Bevor dieses Programm nun gestartet wird, speichert man es sinnvollerweise auf einer Diskette, so daß es hinfort zur Verfügung steht und nicht bei jeder Benutzung zuerst assembliert werden muß.

Allerdings muß dieses Programm als Binärdatei gespeichert werden, weil es ja binär codiert vorliegt - gerade das macht ja den Vorteil der Benutzung von ASSEMBLER-Unterprogrammen (genauer von assemblierten Unterprogrammen) aus: Weil sie binär codiert vorliegen, können sie beim Programmlauf später sehr viel rascher abgearbeitet werden als etwa ein normales BASIC-Programm.

Die binäre Speicherung erfordert es, die Länge des Programms zu kalkulieren:

$$\begin{aligned} \text{Länge} &= \text{Endadresse} - \text{Anfangsadresse} + 1 \\ &= 3\emptyset9 - 3\emptyset\emptyset + 1 \\ &= 1\emptyset = A_{(\text{hexadez.})} \end{aligned}$$

Wir erkennen, daß Anfangs- und Endadresse aus der assemblierten Liste entnommen werden. Es handelt sich dabei um die Satznummern der End- und der Anfangszeile des "echten" Objektcodes. Dieser beginnt ja, wie schon ausgeführt wurde, in der Zeile mit dem Etikett ERROR und endet mit der Anweisung RTS (alle anderen Zeilen beinhalten nicht ausführbare Anweisungen (;) bzw. Direktiven).

Das Speichern des Objektprogramms erfordert also folgendes Kommando:

```
BSAVE FEHLER.OBJ,A8ØØ,LA
                Kenn.      Länge(hex.)
        Name          Startadr.(hex.)
```

(Dieses Kommando muß beim LISA-ASSEMBLER mit den Tasten CTRL und D eingeleitet werden).

Soll nun dieses Programm benutzt werden, so benötigen wir nur das Kommando:

```
BLOAD FEHLER.OBJ,A3ØØ          Startadresse des Objekt-
                               programms (hex.) gemäß
                               ORG-Anweisung
```

Wie ein derartiges ASSEMBLER-Programm benutzt, d.h. in BASIC-Programme "eingebaut" werden kann, darüber soll in den folgenden Abschnitten gesprochen werden.

Ein tieferes Eindringen in die Geheimnisse der ASSEMBLER-Programmierung soll hier nicht erfolgen.
Es ist ja nicht Ziel dieses Buches, beispielsweise einen einführenden ASSEMBLER-Kurs bereitzustellen, als vielmehr zu zeigen, wie man die Möglichkeiten des APPLE-Rechnert nutzen kann. Unter diesem Gesichtspunkt kommt es jetzt mehr auf die folgenden Abschnitte an und die Beispiele, die oben vorgestellt wurden, sind vielleicht schon etwas zu ausführlich erörtert worden.
Gleichwohl ist es sicherlich nicht ganz unwichtig, zumindest in groben Umrissen erfahren zu haben, wie ein ASSEMBLER-Programm aufgebaut werden kann und was unter "Assemblierung" überhaupt zu verstehen ist.

3.2.2 Verknüpfungen BASIC-ASSEMBLER

Auch ohne einen eigenen Assembler zu erwerben, wie z.B. das im vorangegangenen Abschnitt erwähnte Programm LISA, können mit dem APPLE II einige Möglichkeiten der ASSEMBLER-Programmierung genutzt werden, und zwar mit Hilfe des sog. Monitors.

Dieser Monitor ist ein zur Verfügung stehendes Hilfsprogramm, das nützliche Dienste - insbesondere in Verbindung mit BASIC-Programmen - leisten kann.
Die Verbindung, die bei Einbau eines ASSEMBLER-Programms zwischen diesem und dem aufrufenden BASIC-Programm hergestellt werden muß, besorgt dieser Monitor.

Was der Monitor im einzelnen alles zu leisten vermag, soll hier kurz skizziert werden, nachdem zunächst erläutert wird, wie dieses Dienstprogramm aufgerufen wird.

Der Monitor wird mittels einer Anweisung aufgerufen, die wir bisher noch nicht besprochen haben, der CALL-Anweisung:

```
nn  CALL  m
```

Wenn diese Anweisung bei der Abarbeitung eines BASIC-Programms erreicht wird, so erfolgt eine Programmverzweigung in ein Unterprogramm, welches im Gegensatz zu

einem BASIC-Unterprogramm im Maschinensprache geschrieben ist. Die Zahl m im CALL-Statement gibt dabei die Startadresse des jeweiligen Maschinen-Unterprogramms an.

Derartige Unterprogramme befinden sich in Festwertspeichern (also nicht löschbar) im Anfangsadressenbereich zwischen

	DØØØ	und	FFFF	(hexadez.)
also	53248	und	65535	(dezimal)

Die erste Anweisung des eben erwähnten Monitorprogramms befindet sich bei der Adresse FF69 (= 65385 dezimal). Im CALL-Statement muß für die Zahl m die dezimale Schreibweise benutzt werden, wobei anzumerken ist, daß allen Speicherplätzen, deren Adressen größer sind als 32767 (dezimal), auch negative Adressen zugeordnet sind.

Diese negative Adresse erhält man, indem man von der positiven Adresse die Gesamtzahl der Adressen (= 65536) subtrahiert.

Da die Startadresse des Monitors also 65385 ist, kann sie auch mit

65385 - 65536 = -151

bezeichnet werden.

Zum Aufruf des Monitors benötigen wir also die folgende BASIC-Anweisung:

CALL -151

Das Markierungszeichen für Applesoft (]) wird nach Ausführung dieser Anweisung ersetzt durch * , um dem Benutzer anzuzeigen, daß er jetzt auf der maschinensprachlichen Ebene operiert.

Will man wieder auf die "BASIC-Ebene" zurück, so ist die Tastenkombination

[CTRL] und [C]

und anschließend die Return-Taste zu betätigen.
Es erscheint dann wieder das Applesoft-Zeichen].

Was kann nun mit dem Monitor, wenn er über CALL -151 geladen wurde, angefangen werden?

Zunächst kann dieses Dienstprogramm dazu benutzt werden, Speicherinhalte auszugeben.

Geben wir ein:

CALL -151

und nach dem erschienenen Stern z.B. die Adresse 11FF, so antwortet der Monitor:

11FF- 24

d.h. in der Speicherstelle 11FF (hex.) befindet sich der Wert 24 (hex.), also der Wert 36 (dez.).

Wenn man nun einfach nur die die Return-Taste betätigt, gibt der Rechner die Inhalte weiterer Speicherplätze aus, die der Adresse 11FF folgen.

In jeder Zeile, die per Return-Tastenbetätigung entsteht, stehen maximal acht Speicherinhalte, und zwar genau dann acht, wenn die Adresse durch 8 teilbar ist.

Gibt man beispielsweise ein

ØØFF

so antwortet der Rechner mit

ØØFF- 1Ø

und nach Betätigung der Return-Taste erscheinen acht Werte

Ø1ØØ- 31 35 31 ØØ 3Ø 3Ø 3Ø 3Ø weil
Ø1ØØ (hex.)= 256 (dez.) durch 8 teilbar ist.

Will man noch mehr Speicherplätze gleichzeitig inspizieren, so kann man zwei Adressen gleichzeitig eingeben, die durch Punkt zu trennen sind.

Ausgegeben werden dann alle Speicherinhalte zwischen diesen beiden Adressen unter Einschluß der beiden.

Beispiel:

F8ØØ.F83F

Mit dem Monitor können aber nicht nur Speicherinhalte festgestellt, sondern Speicherinhalte auch verändert werden.

Geben wir beispielsweise ein:

FF

so antwortet der Monitor, wie oben beschrieben, mit

ØØFF - 1Ø

Geben wir nun zusätzlich ein

:Ø9

so wird jetzt in der Speicherstelle FF der Wert 9 gespeichert, was wir überprüfen können, wenn wir erneut eingeben

FF

Der Monitor antwortet jetzt:

ØØFF - Ø9

Der Monitor kann aber mehr als nur Speicherinhalte inspizieren oder verändern (wobei natürlich pro Speicherstelle keine Zahl untergebracht werden kann, die größer als FF (hex.) = 255 (dez.) ist).
Die zusätzlichen Möglichkeiten sollen aber nur kurz erwähnt werden:

Von praktischer Bedeutung dürfte ab und zu die Möglichkeit sein, daß innerhalb des Arbeitsspeichers mit Hilfe des Monitors ganze Bereiche kopiert werden können.

Geben wir z.B. nach Aufruf des Monitors die Anweisung

```
12ØØ < 2ØØØ.21ØØM
```

ein, so werden die Inhalte der Speicherplätze 2ØØØ bis 21ØØ in den Speicherbereich übertragen (kopiert), der mit der Adresse 12ØØ beginnt.
Es versteht sich, daß auf diese Weise Datenaustauschprozesse, weil sie auf der maschinensprachlichen Ebene vollzogen werden, schneller ablaufen, als wenn dieser Austausch innerhalb eines BASIC-Programms mit LET-Statements (z.B. innerhalb einer FOR...NEXT-Schleife) durchgeführt würde.

Allerdings macht es die Nutzung solcher Beschleunigungsmöglichkeiten erforderlich, daß der Benutzer den Überblick über die Speichereinteilung besitzt und behält: Er muß wissen, wo was steht und wohin was transferiert werden soll.

Es leuchtet ein, daß auf diese Weise auch z.B. das Löschen von Speicherbereichen schneller vonstatten gehen kann, als wenn man per LET-Statements alle interessierenden Felder mit Nullen belegen wollte.

Man braucht dazu nur diejenige Adresse A, die direkt vor dem zu löschenden Bereich liegt, mit Ø zu besetzen und gibt dann nach obigem Muster an, daß von A+1 beginnend der Bereich A bis A+X kopiert werden soll.

In die Stelle A+1 wird daraufhin die Ø aus der Stelle A kopiert, in die Stelle A+2 die Ø, die inzwischen ja auch in A+1 steht...usw.

Nach formal dem gleichen Muster funktioniert die Monitor-Anweisung zum Vergleichen von Speicherbereichsinhalten. An der Stelle, an der in der obigen Anweisung der Buchstabe M steht, steht jetzt allerdings ein V.

Stellt der Monitor bei derartigen Vergleichen eine Nichtübereinstimmung fest, so meldet er dies.

Eine derartige Vergleichsprozedur kann sehr hilfreich sein, wenn es z.B. um die Prüfung geht, ob eine Datei, die auf Diskette übertragen wurde, korrekt übertragen wurde, oder ob Übertragungsfehler aufgetaucht sind. Eine derartige Prüfung nennt man Verifizierung.

Eine solche Verifizierung könnte folgendermaßen vor sich gehen:

Nehmen wir an, eine Datei bestehe aus Daten, die 1ØØ (dez.) Bytes im Arbeitsspeicher belegen. Belegt seien damit die Speicherplätze 2ØØØ bis 2Ø64 (hex.). Um diese Informationen auf Diskette zu speichern, müssen wir zurück in den BASIC-Modus, geben also

CTRL - C und Return-Taste

gemeinsam ein und danach das Kommando zum Speichern:

BSAVE DATEN,A2ØØØ,L64

(es muß der Befehl zum Speichern einer Binärdatei benutzt werden, also BSAVE und nicht etwa SAVE, und Adressen- und Feldlängenangaben sind hexadezimal anzugeben).

Eine Überprüfung des Speicherungsprozesses kann nun so erfolgen, daß die gerade "weg-"gespeicherte Datei wieder eingelesen wird, und zwar in einen jetzt freien Speicherbereich (wir sehen hier wieder, daß der Benutzer einen Speicherüberblick haben muß bzw. notfalls Speicherbereiche inspizieren und ggf. auf Null setzen muß, wie oben beschrieben).

Nehmen wir an, der Bereich ab Adresse 21ØØ (hex.) sei frei, so können wir eingeben:

BLOAD DATEN,A21ØØ

Mit dem Monitor kann dann verglichen werden:

```
CALL -151
2ØØØ < 21ØØ.2164V
```

Gibt der Monitor keine Fehlermeldung aus, so ist die Verifizierung gelungen, andernfalls müßte erneut gespeichert und verifiziert werden.

Das Monitorprogramm erlaubt auch Verzweigungen. Dazu muß eine Adresse angegeben werden, gefolgt von dem Buchstaben G.

Beispiel:

```
CALL -151
3DØG
```

Diese Anweisung bewirkt einen Sprung in die Maschinensprachanweisung mit der Adresse 3DØ (hex.).

An dieser Adresse steht die Maschinenanweisung

```
JMP 9DB9
```

JMP steht für "Sprung" (Jump) und die Adresse 9DB9 kennzeichnet das Maschinenprogramm, welches den Applesoft-Compiler von einer Diskette einliest.

Damit ist die Verwendung hergestellt zu den Ausführungen im Abschnitt zuvor, wo über ASSEMBLER-Anweisungen gesprochen wurde - JMP ist nichts anderes als eine derartige Anweisung.

Dies verdeutlicht, daß innerhalb des Monitors ein Assembler zur Verfügung steht, dem wir uns nun zuwenden wollen. Es muß dabei darauf hingewiesen werden, daß dies nur eine sehr bescheidene ASSEMBLER-Version ist, die nur für einen beschränkten Kreis von Aufgaben eingesetzt werden kann.

Die Startadresse dieses Assemblers ist F666 (hex.) = -2458 (dez.). Aufrufbar ist er also wie folgt:

```
CALL  -151
F666G
```

oder direkt (ohne den Monitor einzuschalten):

```
CALL  -2458
```

Dieser Assembler steht allerdings nur zur Verfügung, wenn der APPLE II mit der Steckkarte INTEGER-BASIC ausgestattet ist, oder wenn es sich um einen Standard-APPLE II handelt.

Ist der Assembler geladen, meldet er sich mit einem Ausrufungszeichen als Markierung (anstelle von > für INTEGER-BASIC bzw. * für den Monitor).

Die Assembler-Anweisungen werden hinter dem ! genauso formuliert wie die Monitor-Anweisungen hinter dem * , allerdings ist zunächst das Dollarzeichen ($) einzugeben.

Beispielsweise kann eine Speicherstelle inspiziert werden, wenn hinter dem !-Zeichen eingegeben wird:

$1CFF

Der Assembler antwortet mit

1CFF -6E

Die Rückkehr von der Assembler-Ebene auf die Monitorebene erfolgt mit der Anweisung FF69G, der ein $-Zeichen vorangestellt werden muß, also

$FF69G

Entsprechend gelangen wir mit

$ CTRL und C

auf die Applesoft-Ebene zurück.

Die einzelnen ASSEMBLER-Anweisungen, die bei diesem Mini-Assembler möglich sind, sollen hier nicht besprochen werden, weil ja - wie schon erwähnt - dieser Assembler nicht bei allen APPLE-Versionen zur Verfügung steht.
Wir verweisen dazu auf die zuständige Spezialliteratur (siehe z.B. Lon Poole, Martin McNiff und Steven Cook: Apple II Anwenderhandbuch, München 1981, S. 7-16 ff.)

3.3 Unterprogramme in Maschinensprache

Im Festwertspeicher des APPLE II befinden sich eine Reihe von Unterprogrammen in Maschinensprache, die man bei bestimmten Aufgabenstellungen sehr nutzbringend einsetzen kann, wie im folgenden gezeigt werden soll.

Diese Unterprogramme braucht der Benutzer also nicht selbst etwa in ASSEMBLER zu programmieren, sondern sie können mit der schon besprochenen CALL-Anweisung aufgerufen werden. Sollte der Benutzer aber über ein Assembler-Programm, wie z.B. das oben erwähnte Programm LISA verfügen, kann er natürlich, wie im vorangegangenen Abschnitt ausgeführt wurde, diese vorhandenen Unterprogramme um eigene Entwicklungen ergänzen.

Die wichtigsten dieser Unterprogramme beziehen sich auf die folgenden Aufgabenstellungen:

1. Graphische Darstellungen in Normalgraphik;
2. Tastatureingaben;
3. Textausgaben;
4. Ausgabe akustischer Signale;
5. Cursor-Steuerung;
6. Veränderungen auf dem Bildschirm.

Zu einigen dieser Aufgabenstellungen sollen nun kurze Beispiele vorgestellt werden, um die Funktionsweise unterschiedlicher Unterprogramme zu illustrieren:

Beispiel 1: Erzeugen von Tönen

Das folgende Programm erzeugt einige Piepstöne unter Benutzung eines Unterprogramms, das über

```
CALL  -1Ø52
```

aufgerufen wird.

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 5)"3.11 - UNTERPROGRAMM PIEPSEN"
60  GOSUB 1000: REM  WARTEN
70  FOR I = 1 TO 15
80  PRINT I;". MAL  :  PIEPS"
90  CALL  - 1052
100  FOR J = 1 TO 99: NEXT J
110  NEXT I
120  PRINT : PRINT : PRINT "ENDE DER AUSGABE": END
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!": NORMAL
1020  GET A$: IF A$ = "" THEN 1020
1030  HOME : RETURN
```

Programmbeschreibung

Sätze	Inhalt
3Ø- 6Ø	Überschrift und Warten
7Ø-11Ø	Es wird 15 mal gepiepst (mit Kontrollausdruck) 1ØØ: Zwischen den Tönen jeweils kurze Warteschleifen
12Ø	Ende des Programms
1ØØØ-1Ø3Ø	Unterprogramm Warten 1Ø1Ø: Meldung 1Ø2Ø: Abwarten, bis im Feld A$ etwas empfangen wird (Tastendruck) 1Ø3Ø: Löschen des Bildschirms und Rücksprung

Beispiel 2: Verschiebung des Cursors

Das folgende Programm zeigt, wie der Cursor nach oben verschoben werden kann. Dazu benutzen wir ein Unterprogramm, das über

```
CALL -998
```

aufgerufen wird.

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 7)"3.12 - CURSOR-VERSCHIEBUNG"
60  GOSUB 1000: REM  WARTEN
70  FOR I = 1 TO 5
80  PRINT I;". AUSGABE"
90  NEXT I
100  PRINT : PRINT : PRINT "ES FOLGEN 5 LEERZEILEN"
110  PRINT : PRINT : PRINT : PRINT : PRINT : PRINT "DAS WAREN
SIE"
120  PRINT : PRINT "NACH DEM CONT WIRD DER CURSOR 12 ZEILEN"
130  PRINT "NACH OBEN VERSETZT, WO DANN **AHA** GE-"
135  PRINT "DRUCKT WIRD": PRINT : PRINT "GIB CONT EIN": STOP

140  FOR I = 1 TO 12
150  CALL  - 998
160  NEXT I
165  FLASH
170  PRINT  TAB( 15)"**  AHA  **              "
175  NORMAL
180  VTAB 23: PRINT "ENDE DER AUSGABE": END
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!": NORMAL
1020  GET A$: IF A$ = "" THEN 1020
1030  HOME : RETURN
```

Programmbeschreibung

Sätze	Inhalt
3Ø- 6Ø	Überschrift und Abwarten (UP 1ØØØ)
7Ø- 9Ø	Fünfmalige Angabe des Worts "AUSGABE"
1ØØ	Hinweis auf fünf folgende Leerzeilen
11Ø	Fünf Leerzeilen
12Ø-135	Erläuternde Hinweise und Unterbrechung des Programmablaufs
14Ø-16Ø	Verschiebung des Cursors um 12 Zeilen nach oben
165-175	Ausgabe des Worts "AHA" in blinkender Schrift (Einzelheiten zu FLASH und NORMAL werden in Kap. 4 besprochen)
18Ø	Beendigung des Programms nach entsprechender Meldung in Zeile 23.
1ØØØ-1Ø3Ø	Unterprogramm Warten 1Ø1Ø: Meldung in Zeile 23 ab Spalte 6 in inverser Schrift und Rückkehr zur normalen Schreibweise 1Ø2Ø: Abwarten einer Tasteneingabe 1Ø3Ø: Rücksprung

Beispiel 3: Fehlermeldung

Das folgende Programm demonstriert die Benutzung eines Maschinenprogramms, das über

```
CALL -211
```

aufgerufen wird, zur Fehlermeldung durch optische und akustische Anzeige:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 10)"3.13 - FEHLERMELDUNG"
60  GOSUB 1000: REM  WARTEN
70  HOME
80  PRINT "ES FOLGT NUN EINE FEHLERMELDUNG : ": PRINT : PRINT
: PRINT
90  CALL  - 211
95  PRINT : PRINT : PRINT : PRINT : PRINT
100  PRINT : PRINT : PRINT "ENDE": END
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!": NORMAL
1020  GET A$: IF A$ = "" THEN 1020
1030  HOME : RETURN
```

Programmbeschreibung

Sätze	Inhalt
3Ø- 6Ø	Überschrift und Abwarten (UP 1ØØØ)
7Ø- 8Ø	Löschen des Bildschirms und Ausgabe eines erläuternden Hinweises.
9Ø	Aufruf des Unterprogramms, welches das Wort ERR (=error =Fehler) und einen Piepston ausgibt
95-1ØØ	Beendigung des Programms
1ØØØ-1Ø3Ø	Unterprogramm Warten (Details siehe Beispiel 2)

3.4 Anwendungsbeispiele

Es wurde schon in vielen Programmbeispielen ein Unterprogramm benutzt, welches dazu diente, den Programmablauf zu unterbrechen, so daß der Rechner so lange mit weiteren Aktivitäten (z.B. Bildschirmausgaben) wartete, bis der Benutzer irgendeine Taste betätigte.

Dieses Unterprogramm sieht folgendermaßen aus:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 8)"3.14 - WARTEN/VERSION 1"
60  PRINT : PRINT : PRINT : PRINT : PRINT "DIESES PROGRAMM WARTET
EINE TASTEN-   "
70  PRINT "EINGABE DURCH SPRUNG INS UNTERPROGRAMM"
80  PRINT "1000 AB : "
90  GOSUB 1000: REM  WARTEN
100  PRINT "ICH HABE GEWARTET": PRINT : PRINT
110  PRINT "NACHDEM DIE TASTENEINGABE ERFOLGT IST"
120  PRINT "WIRD DAS PROGRAMM NUN BEENDET": END
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE
1020  PRINT "BITTE EINE TASTE DRUECKEN !"
1030  NORMAL
1040  GET A$: IF A$ = "" THEN 1040
1050  HOME : RETURN
```

In diesem Unterprogramm geschieht nach dem nicht-ausführbaren REM-Statement (1ØØØ) folgendes:

1. In Satz 1Ø1Ø wird ein inverser Ausdruck (INVERSE bewirkt Ausgabe "schwarz auf weiß" auf dem Bildschirm) in Zeile 23 (VTAB 23) ab Spalte 6 (HTAB 6) vorbereitet.

2. In Satz 1Ø2Ø erfolgt diese so vorbereitete Ausgabe.

3. In Satz 1Ø3Ø wird auf die Normalschreibweise zurückgeschaltet ("weiß auf schwarz", auf dem Bildschirm per Anweisung NORMAL).

4. In Satz 1Ø4Ø wird eine Tastatureingabe erwartet. Solange diese nicht erfolgt,d.h. solange A$="" (im Feld A$ ist (noch) nichts), solange wird durch Sprung nach 1Ø4Ø weitergewartet.

5. Ist hingegen eine Tasteneingabe erfolgt, geht es weiter bei Satz 1Ø5Ø, d.h. der Bildschirm wird gelöscht (HOME) und es erfolgt der Rücksprung ins Hauptprogramm (RETURN).

In einigen der vorangegangenen Beispiele haben wir diese insgesamt 1Ø Statements in noch weniger als sechs Sätzen zusammengepackt.

Ein derartiges Warteprogramm - und deshalb wurde es hier erwähnt - kann nun auch sehr elegant mit Hilfe von Maschinen-Unterprogrammen erstellt werden. Dadurch läuft es schneller ab und man vermeidet zudem das GET-Statement, welches bei Verwendung im Zusammenhang mit hochauflösender Graphik (vergl. Kap. 4) manchmal sehr störende Fehler bei der Graphikausgabe verursacht.

Alternative Programmierung des "Warteprogramms":

```
30   PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40   PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50   PRINT : PRINT : PRINT  TAB( 8)"3.15 - WARTEN/VERSION 2"
60   PRINT : PRINT : PRINT : PRINT : PRINT "DIESES PROGRAMM WARTET
EINE TASTEN-    "
70   PRINT "EINGABE DURCH SPRUNG INS UNTERPROGRAMM"
80   PRINT "1000 AB : "
90   GOSUB 1000: REM  WARTEN
100   PRINT "ICH HABE GEWARTET": PRINT : PRINT
110   PRINT "NACHDEM DIE TASTENEINGABE ERFOLGT IST"
120   PRINT "WIRD DAS PROGRAMM NUN BEENDET": END
1000   REM  UP WARTEN
1010   VTAB 23
1020   CALL  - 868
1030   INVERSE : HTAB 6
1040   PRINT "BITTE EINE TASTE DRUECKEN !"
1050   NORMAL
1060   POKE  - 16368,0
1070   WAIT  - 16384,128
1080   POKE  - 16368,0
1090   VTAB 23
1100   CALL  - 868
1110   RETURN
```

Auch dieses Programm kann natürlich enger geschrieben werden; im Prinzip kann man es zwei Sätze verkürzen:

Satz 1ØØØ : Statement 1ØØØ

Satz 1Ø1Ø : Statements 1Ø1Ø-111Ø.

Wir haben es deshalb so "ausführlich" oben programmiert, um für die folgende Programmbeschreibung schon eine praktische Numerierung zur Hand zu haben.

Programmbeschreibung

Sätze	Inhalt
1ØØØ	Kommentar (als REM-Statement nicht ausführen)
1Ø1Ø	Auswahl von Bildschirmzeile 23
1Ø2Ø	Löschen dieser Zeile per Maschinenprogramm
1Ø3Ø	Umschalten auf inversen Schreibmodus und Beginn bei Bildschirmspalte 6
1Ø4Ø	Textausgabe
1Ø5Ø	Zurückschalten auf Normalschreibweise
1Ø6Ø	Löschen der Speicherstelle -16368
1Ø7Ø	Abwarten einer Tastatureingabe (ASCII-Codezahl 128 oder größer)

Sätze	Inhalt
1Ø8Ø	Erneutes Löschen der Speicherstelle -16368, nachdem die Tastatureingabe erfolgt ist, damit mit dieser Inputinformation nichts angefangen werden kann.
1Ø9Ø-11ØØ	Löschen der Bildschirmzeile 23
111Ø	Rücksprung

In einem weiteren Beispiel wollen wir die Möglichkeit der Cursorverschiebung nutzen, die im vorangegangenen Abschnitt vorgestellt wurde.

In vielen Programmen wird der Benutzer beispielsweise dazu aufgefordert, mit J (für "Ja") oder N (für "Nein") auf eine durch den Rechner gestellte Frage zu antworten. Es wäre nun ganz sinnvoll, prinzipiell zu prüfen, ob der Benutzer irrtümlicherweise ein anderes Symbol als J oder N eingegeben hat. Ist dies der Fall, sollte ein Piepston akustisch warnen und der Cursor müßte an der Stelle stehen bleiben, an der die korrekte Eingabe (eben nur J oder N) erwartet wird.

Weiterhin wäre es sinnvoll, wenn nach korrekter Eingabe (also J-Taste oder N-Taste) das Programm ohne Aufenthalt weiterarbeiten würde.

Hat man eine solche Prüfung im Programm nicht, so könnte es folgendermaßen aussehen:

```
100  HOME
110  PRINT "SOLL ICH WEITERARBEITEN (JA ODER NEIN)?"
115  PRINT : PRINT : PRINT
120  INPUT "GIB J ODER N EIN : ";A$
130  IF A$ = "N" THEN 110
140  PRINT : PRINT : PRINT "ES GEHT WEITER"
150  REM  WEITERE DATENVERARBEITUNG
160  PRINT : PRINT : PRINT "ENDE": END
```

Dieses Programm reagiert auf die Eingabe von J korrekt. Wird hingegen eine falsche Taste betätigt, sc wird diese Eingabe behandelt wie eine Ja-Antwort - und dies ist kein erfreulicher Zustand.

Besser funktioniert das folgende Programm:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT, TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 3)"3.16 - KORREKTUR FALSCHER
EINGABEN"
60  GOSUB 1000: REM  WARTEN
70  HOME
80  PRINT "SOLL DAS PROGRAMM WEITERARBEITEN ?"
90  PRINT : PRINT : PRINT
100  PRINT "BITTE J ODER N EINGEBEN ";: GET A$: PRINT : IF A$
<  > "J" AND A$ <  > "N" THEN  CALL  - 1052: CALL  - 998: GOTO
100
120  IF A$ = "N" THEN  PRINT : PRINT : PRINT : GOTO 80
130  PRINT : PRINT : PRINT : PRINT "DAS PROGRAMM KANN JETZT
WEITERGEHEN"
140  REM  WEITERE DATENVERARBEITUNG
150  PRINT : PRINT : PRINT : PRINT "ENDE": END
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!": NORMAL
1020  GET A$: IF A$ = "" THEN 1020
1030  HOME : RETURN
```

Die Programmbeschreibung zeigt, worum es geht:

Sätze	Inhalt
3Ø- 5Ø	Überschrift
6Ø	Abwarten durch Sprung ins Unterprogramm 1ØØØ
7Ø	Löschen des Bildschirms
8Ø	Frage, ob das Programm weiterarbeiten soll (=Testfrage für dieses Programm)
9Ø	Leerzeilen
1ØØ	Hinweis, daß J oder N eingegeben werden soll; Aufnahme der Eingabe durch die GET-Anweisung; Leerzeile; Wenn im Feld weder "J" noch "N" steht (fehlerhafte Eingabe!), dann wird ein Piepston als akustische Warnung erzeugt (CALL -1Ø52); danach wird der Cursor eine Zeile zurückgesetzt und der Hinweis wiederholt (Rücksprung an den Satzanfang 1ØØ)
12Ø	Ist "N" eingegeben worden, wird die Frage aus Satz 8Ø wiederholt (Rücksprung nach 8Ø); genauso gut könnten natürlich jetzt irgendwelche alternativen Berechnungen oder Verarbeitungsprozesse durchgeführt werden.
13Ø-14Ø	Weiterarbeiten im Programm; bzw. Durchführung anderer Datenverarbeitungsprozesse
15Ø	Beendigung des Testprogramms

Sätze	Inhalt
1ØØØ-1Ø3Ø	Unterprogramm Warten (an anderer Stelle schon erläutert)

In einem letzten Beispiel zeigen wir, wie vorgegangen werden kann, um fehlerhafte (beispielsweise numerische) Angaben in geschickter Weise zu korrigieren:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 5)"3.18-KORREKTUR FALSCHER DATEN"
60  GOSUB 1000: REM  WARTEN
70  HOME
80  PRINT "BITTE GEBEN SIE IN DER FOLGENDEN ZEILE"
90  INPUT "EINE DEZIMALZAHL EIN : ";X
100  PRINT : PRINT : PRINT
110  GOSUB 3000: REM  KORREKTUR
120  PRINT : PRINT : PRINT "DAS PROGRAMM KANN JETZT WEITERARBEITEN"
130  REM  WEITERE DATENVERARBEITUNG
140  PRINT : PRINT : PRINT,"ENDE": END
150  PRINT : PRINT : PRINT : PRINT "ENDE": END
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!": NORMAL
1020  GET A$: IF A$ = "" THEN 1020
1030  HOME : RETURN
3000  REM  UP KORREKTUR
3010  GOSUB 6000: REM  PRUEFEN
3020  IF A$ = "J" THEN  RETURN
3030  PRINT : PRINT "FALSCHE ANGABE : ";X
3040  PRINT : PRINT : INPUT "KORREKTUR       : ";X
3050  GOTO 3010
6000  REM  UP PRUEFEN
6010  PRINT "IST DIE EINGABE KORREKT ?(J/N)";: GET A$: PRINT
: IF A$ <  > "J" AND A$ <  > "N" THEN  CALL  - 1052: CALL  -
998: GOTO 6010
6020  CALL  - 998: CALL  - 868: RETURN
```

Auch hier erläutert die Programmbeschreibung, was im einzelnen geschieht:

Sätze	Inhalt
3Ø- 5Ø	Überschrift
6Ø	Abwarten durch Sprung ins Unterprogramm 1ØØØ
7Ø	Löschen des Bildschirms
8Ø- 9Ø	Anforderung einer Dezimalzahl zu Testzwecken
1ØØ	Leerzeilen
11Ø	Sprung ins Unterprogramm 3ØØØ, um die eingegebene Zahl (8Ø-9Ø) zu prüfen und gegebenenfalls zu korrigieren
12Ø-13Ø	Nach erfolgtem Rücksprung kann das Programm mit weiteren Datenverarbeitungsprozessen fortgeführt werden
14Ø	Beendigung des Programms
1ØØØ-1Ø3Ø	Unterprogramm Warten (an anderer Stelle schon erläutert)

Sätze	Inhalt
3ØØØ-3Ø5Ø	Unterprogramm zur Korrektur fehlerhaft eingegebener Werte: 3Ø1Ø: Sprung ins Unterprogramm 6ØØØ zur Überprüfung; 3Ø2Ø: Wird mit der Eingabe "J" (für Ja) im Unterprogramm 6ØØØ die Korrektheit der Testzahl bestätigt, erfolgt der Rücksprung ins Hauptprogramm; 3Ø3Ø: Ist die Testzahl hingegen falsch, wird sie ausgegeben 3Ø4Ø: Danach wird die korrigierte Zahl angefordert 3Ø5Ø: Rücksprung nach 3Ø1Ø, um auch diese Neueingabe (3Ø4Ø) zu überprüfen...usw.
6ØØØ-6Ø2Ø	Unterprogramm zur Prüfung eingegebener Werte: 6Ø1Ø: Abfrage, ob der eingegebene Wert korrekt ist; Überprüfung der Eingaben "J" (für Ja), bzw. "N" (für Nein) (vergl. vorangegangenes Beispiel) 6Ø2Ø: Herstellung der Ausgangslage und Rückkehr (mit korrektem "J" oder "N") ins Unterprogramm 3ØØØ (Satz 3Ø2Ø)

Diese Beispiele mögen genügen, um ansatzweise zu zeigen, wie insbesondere bei einer dialogorientierten Programmierung durch Verwendung geeigneter Maschinenprogramme die Programme wesentlich verbessert und komfortabler gemacht werden können.

Kapitel 4: Graphikprogramme

4.1 Vorbemerkungen

Die Erstellung von Graphiken mit dem Computer ist eine der reizvollsten Aufgaben überhaupt - und eine wichtige Aufgabe dazu: Man denke beispielsweise an die zunehmende Bedeutung des sog. Computer-Konstruierens, bei dem der Rechner schon weitgehend die früher üblichen, überdimensional großen Zeichenbretter ersetzt hat.

In der Art und Weise, wie die verschiedenen Rechner Graphikanforderungen der Benutzer und ihren Wünschen gerecht werden können, unterscheiden sie sich voneinander: Es gibt graphikfähige und "unfähige" Computer; es gibt Rechner, die erst mit Hilfe mehr oder weniger umfangreicher (und kostspieliger) Software graphikfähig werden, aber auch solche, die "von Hause aus" graphikfähig sind.

Der APPLE-Rechner, um den es hier geht, gehört zur letzten Kategorie.
Von vornherein muß berücksichtigt werden, daß der Apple, wie einige andere Rechner auch, zwei Arten der graphischen Darstellung unterscheidet:

1. Die sog. Normalgraphik
2. Die hochauflösende Graphik

Im ersten Fall muß man sich den Bildschirm aufgeteilt denken in 4Ø Spalten und 48 Zeilen und jeder einzelne der so entstehenden "Kreuzungsbereiche" ist einzeln ansprechbar.

Von den 48 Zeilen stehen im Graphik-Modus per Voreinstellung (von der abgewichen werden kann) nur 44 Zeilen zur Verfügung (Ø bis 43), während der restliche Platz für zwei Textzeilen am unteren Bildschirmrand reserviert ist.

Jeder dieser Kreuzungsbereiche ist aber vergleichsweise groß (auf einem 3Ø-cm-Bildschirm ca. 3 * 5 mm), so daß Graphiken, die auf dieser Grundlage aufgebaut werden, relativ "grob" sind.

Dies ist anders bei der hochauflösenden Graphik:

Hier hat man sich den Bildschirm aufgeteilt zu denken in 192 Zeilen und 28Ø Spalten, so daß jetzt insgesamt 5376o "Kreuzungspunkte" entstehen, von denen jeder einzelne ansprechbar ist. Dies ermöglicht sehr viel "feinere" graphische Darstellungen - eben "höher auflösende" Graphiken.

4.2 Normalgraphik

Die Normalgraphik ist die nicht so sehr interessante Form der Möglichkeit, graphische Darstellungen über BASIC-Programm zu erzeugen.
Gleichwohl ist sie für spezielle Fragestellungen, wie sich in späteren Beispielen zeigen wird, sehr gut geeignet.

4.2.1 Textmodus und Graphikmodus

Normalerweise befindet sich der Bildschirm im Textmodus, d.h. wenn wir ein Programm eintippen, dann erscheinen die Programmzeilen, d.h. die gesamte Programmliste, als lesbarer Text auf dem Bildschirm (glücklicherweise). Weiterhin tauchen auch Ergebnisse, die über das PRINT-Statement ausgegeben werden, als direkt lesbare Sachverhalte auf dem Bildschirm auf.

Sollen nun beispielsweise Graphiken ausgegeben werden, so muß zunächst in den Graphikmodus "umgeschaltet" werden.

Diese Umschaltung erlaubt die Anweisung

GR	(GR = "Graphic")

Diese Anweisung kann sowohl als Kommando, wie auch als Statement innerhalb eines BASIC-Programms (dann natürlich versehen mit einer Satznummer) benutzt werden.

Wird bei der Programmabarbeitung dieses Statement erreicht, so wird auf den Graphikmodus umgeschaltet, d.h. vorher auf dem Bildschirm vorhandene Texte (z.B. die noch zu sehende Programmliste) verschwinden jetzt. Allerdings werden die untersten beiden Bildschirmzeilen für Texte nach wie vor freigehalten.

Wenn im weiteren Programm nun geeignete Graphik-Anweisungen folgen, wird auf dem Bildschirm das entsprechende Bild erscheinen.
Häufig ist es angebracht, im Anschluß an die Graphikausgabe wieder in den Textmodus zurückzukehren - entweder weil z.B. weitere, nun textliche Ausgaben folgen oder weil z.B. das bisherige Programm verändert oder wegen irgendwelcher Fehler korrigiert oder einfach nur weitergeführt und verlängert werden soll.

Diese Rückkehr in den Textmodus geschieht mit der Anweisung

TEXT

Auch diese Anweisung kann natürlich als BASIC-Anweisung verwendet werden.

Wird sie in der Programmabarbeitung erreicht, verschwindet die Graphik. Es ist deshalb sinnvoll, vor dieser Anweisung, die Anweisung STOP oder GET einzugeben, um dem Programmbenutzer die Möglichkeit zu geben, sich in Ruhe die Graphik anzuschauen.

Zweckmäßigerweise verbindet man dies mit einer Mitteilung etwa von der Form

BITTE CONT EINGEBEN (bei STOP), oder:

BITTE EINE TASTE DRUECKEN (bei GET),

damit auch derjenige, der wenig oder keine Erfahrung mit BASIC-Programmen hat, weiß, was zu tun ist.

Dieser Text kann in eine der beiden unteren Bildschirmzeilen gedruckt werden, ohne daß vorher die Anweisung TEXT erreicht werden muß, denn diese Zeilen stehen ja auf jeden Fall für Textausgaben zur Verfügung.

Zum Ausdruck z.B. in der 23. Zeile benötigen wir beim APPLE die Anweisung

VTAB 23 (VTAB = Vertikaler Tabulator)

4.2.2 Farben und graphische Effekte

Es gibt eine Reihe von Möglichkeiten, schon unabhängig vom Graphikmodus graphische Effekte zu produzieren.

Dies gelingt mit den beiden Statements

```
INVERSE
```

und

```
FLASH
```

Die Anweisung INVERSE "schaltet um" von der Textausgabe "weiß auf schwarz" auf die Textausgabe "schwarz auf weiß".

Mit der Anweisung FLASH wird erreicht, daß die Druckausgabe zwischen beiden Möglichkeiten ständig hin- und her wechselt.

Aus diesen "Sonderdruck"-Ausgaben kann man sich wieder verabschieden durch die Anweisung

```
NORMAL
```

Man erprobe zum Beispiel das folgende Programm:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 10)"4.1 - SONDEREFFEKTE"
60  PRINT : PRINT : PRINT : PRINT : PRINT
70  INVERSE : PRINT "APPLE": NORMAL : PRINT : PRINT
80  FLASH : PRINT "BLITZ": NORMAL
90  PRINT : PRINT : PRINT "ENDE": END
```

Der APPLE leistet aber noch mehr:
Wenn er mit einem Farb-Interface ausgestattet und ein Farbfernseher oder Farbmonitor angeschlossen ist, dann können auch farbige Ausgaben produziert werden.

Dazu benötigen wir die Anweisung

```
COLOR  = Zahl
```

An der Stelle "Zahl" steht dabei ein Wert zwischen Ø und 15 , der die jeweilige Farbe der folgenden Ausgaben bestimmt.

Farbtabelle

Farbe	Zahl	Farbe	Zahl
schwarz	Ø	braun	8
magenta	1	orange	9
dunkelblau	2	grau 2	1o
purpur	3	rosa	11
dunkelgrün	4	hellgrün	12
grau 1	5	gelb	13
mittelblau	6	aquamarinblau	14
hellblau	7	weiß	15

4.2.3 Graphik-Anweisungen

Die Anweisungen, die im einfachen Graphikmodus durchgeführt werden, sollen nun - bevor wir sie an geeigneten Beispielen illustrieren - kurz vorgestellt werden:

Das erste hier infrage kommende Statement lautet

```
PLOT S,Z
```

Es dient dazu, am Kreuzungspunkt der Spalte S mit der Zeile Z einen "Punkt" (in Form eines kleinen Rechtecks). zu zeichnen.

Wir erinnern daran, daß 4Ø Spalten zur Verfügung stehen (sie werden von Ø bis 39 bezeichnet) und für die Graphik 44 Zeilen (Ø bis 43). Deshalb darf an der Stelle S keine negative und keine Zahl größer als 39, an der Stelle Z keine negative und keine Zahl größer als 43 stehen.

Wird diese Regel nicht eingehalten, erscheint folgende Fehlermeldung:

ILLEGAL QUANTITY ERROR IN nn

Allerdings können an den Stellen S und Z reelle Werte (z.B. irgendwelche Rechenergebnisse oder Variablenausprägungen) stehen. Solange sie die obigen Grenzen einhalten, werden sie vom Rechner ganzzahlig gemacht.

Mit diesem Statement können auch Striche auf dem Bildschirm gezeichnet werden. Dazu brauchen wir das PLOT-Statement nur in Schleifen einzubauen, wie das folgende Beispiel zeigt:

```
30   PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40   PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50   PRINT : PRINT  TAB( 7)"4.2 - SENKRECHTER STRICH"
60   GOSUB 1000: REM  WARTEN
70   HOME
80   GR : COLOR= 12: REM  GRUEN
90   FOR I = 5 TO 20
100   PLOT 10,I
110   NEXT I
120   GOSUB 1000: REM  WARTEN
130   TEXT : HOME : PRINT "ENDE": END
1000   REM  UP WARTEN
1010   VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!"
1020   NORMAL
1030   GET A$: IF A$ = "" THEN 1030
1040   RETURN
```

VLIN Z_1, Z_2 AT S

Diese Anweisung erzeugt in der Spalte S des Bildschirms einen senkrechten Strich, der in Zeile Z_1 beginnt und in Zeile Z_2 endet.

Entsprechend zeichnet

HLIN S_1, S_2 AT Z

einen horizontalen Strich in Zeile Z, beginnend in Spalte S_1 und endend in Spalte S_2.

Bei den Werten S, S_1, S_2, Z, Z_1 und Z_2 sind die Grenzen einzuhalten, über die oben schon gesprochen wurde.

4.2.4 Beispiel 1: Bildschirmrand

Für viele Anwendungsprogramme empfiehlt es sich, eine Bildschirmumrandung zu zeichnen. Dafür taugt das folgende Programm:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 14)"4.3 - RAND"
60  GOSUB 1000: REM  WARTEN
70  HOME
80  INPUT "LINKE SPALTE      : ";S1
90  IF S1 < 0 OR S1 > 39 THEN  PRINT : PRINT "FALSCHE EINGABE":
PRINT : GOTO 80
100  PRINT : INPUT "RECHTE SPALTE     : ";S2
110  IF S2 < 0 OR S2 > 39 THEN  PRINT : PRINT "FALSCHE EINGABE":
PRINT : GOTO 100
120  PRINT : INPUT "OBERE ZEILE       : ";Z1
130  IF Z1 < 0 OR Z1 > 43 THEN  PRINT : PRINT "FALSCHE EINGABE":
PRINT : GOTO 120
140  PRINT : INPUT "UNTERE ZEILE      : ";Z2
150  IF Z2 < 0 OR Z2 > 43 THEN  PRINT : PRINT "FALSCHE EINGABE":
PRINT : GOTO 140
160  GR : COLOR= 12: REM  GRUEN
170  HLIN S1,S2 AT Z1
180  HLIN S1,S2 AT Z2
190  VLIN Z1,Z2 AT S1
200  VLIN Z1,Z2 AT S2
210  GOSUB 1000: REM  WARTEN
220  TEXT : HOME : PRINT "ENDE": END
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!"
1020  NORMAL
1030  GET A$: IF A$ = "" THEN 1030
1040  RETURN
```

4.2.5 Beispiel 2: Gitter

Übungshalber soll ein kleines Programm angefügt werden - diesmal ohne Fehlerabfangmechanismen - welches auf dem Bildschirm ein Balkengitter erzeugt. Dem Benutzer bleibt überlassen, die Abstände zwischen den waagrechten und den senkrechten Balken selbst festzulegen.

```
30   PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40   PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50   PRINT : PRINT  TAB( 13)"4.4 - GITTER"
60   GOSUB 1000: REM  WARTEN
70   HOME
80   INPUT "WAAGRECHTER ABSTAND  : ";W
90   PRINT : INPUT "SENKRECHTER ABSTAND  : ";S
100   GR : COLOR= 12: REM  GRUEN
110   FOR I = 0 TO 39 STEP S
120   VLIN 0,39 AT I: NEXT I
130   FOR I = 0 TO 39 STEP W
140   HLIN 0,39 AT I: NEXT I
150   GOSUB 1000: REM  WARTEN
160   TEXT : HOME : PRINT "ENDE": END
1000   REM  UP WARTEN
1010   VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!"
1020   NORMAL
1030   GET A$: IF A$ = "" THEN 1030
1040   RETURN
```

Auch hier erübrigen sich nähere Erläuterungen.

4.2.6 Beispiel 3: Farbdemonstration

Im folgenden Programm sollen in sehr einfacher Form die Farbmöglichkeiten demonstriert werden. Natürlich ist der Einsatz dieses Programms nur sinnvoll, wenn die PAL-Farbkarte des APPLE (Farb-Interface) und ein Farbfernseher oder -monitor zur Verfügung stehen.

```
30   PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40   PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50   PRINT : PRINT  TAB( 13)"4.5 - FARBEN"
60   GOSUB 1000: REM  WARTEN
70   GR
80   FOR F = 0 TO 15
90   COLOR= F
100   HLIN 5,35 AT 2 * F
120   NEXT F
130   GOSUB 1000: REM  WARTEN
140   TEXT : HOME : PRINT "ENDE": END
1000   REM  UP WARTEN
1010   VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!"
1020   NORMAL
1030   GET A$: IF A$ = "" THEN 1030
1040   RETURN
```

Wenn keine Farbgraphik zur Verfügung steht, werden anstelle farbiger Balken Balken mit unterschiedlicher "Schraffur" erzeugt.

4.2.7 Beispiel 4: Histogramm

Ein Histogramm ist das graphische Abbild einer statistischen Häufigkeitsverteilung.

Das entsprechende Graphikprogramm ist also - im Gegensatz zu den bisher vorgestellten Beispielen - das erste direkt anwendungsbezogene bzw. problemlösende Programm.

Dazu benutzen wir das folgende Beispiel:

Altersverteilung in der Bundesrepublik Deutschland 198o:

Altersklasse	Anzahl (in 1ooo)	in %
unter 1o	6317.o	1o.3
1o bis unter 2o	1oo87.7	16.4
2o " " 3o	8964.2	14.6
3o " " 4o	8263.1	13.4
4o " " 5o	8714.o	14.2
5o " " 6o	7332.7	11.9
6o " " 7o	5489.7	8.9
7o " " 8o	48oo.3	7.8
8o " " 9o	1469.2	2.4
9o und mehr	128.5	o,2
Summe	61566.3	1oo.1 *

Quelle: Stat. Jahrbuch der Bundesrepublik Deutschland, 1982, S. 59

* Rundungsungenauigkeiten.

Die in der letzten Spalte der vorstehenden Tabelle angegebenen relativen Häufigkeiten lassen sich durch unterschiedlich lange Balken graphisch darstellen.

Das entsprechende BASIC-Programm könnte folgendermaßen aussehen:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 11)"4.6 - HISTOGRAMM"
60  GOSUB 1000: REM  WARTEN
70 N = 10: REM  KLASSENANZAHL
80  IF N > 39 THEN  PRINT "KLASSENANZAHL ZU GROSS. BITTE KLASSEN":
PRINT "ZUSAMMENLEGEN.": GOTO 500
90  DIM F(N),FT(N)
100  FOR I = 1 TO N: READ F(I): NEXT I
110  REM  MAXIMALER F-WERT
120 FM = 0
130  FOR I = 1 TO N: IF F(I) > FM THEN FM = F(I)
140  NEXT I
150  REM  NORMIERUNG
160  FOR I = 1 TO N:FT(I) = F(I) * (35 / FM)
170 FT(I) =  INT (FT(I) * 100 + .5) / 100: NEXT I
180 SW = 3
200  REM  ZEICHNEN
210  GR : COLOR= 12: REM  GRUEN
220  HLIN 0,39 AT 35
230  FOR I = 1 TO N
240  VLIN 35 - FT(I),35 AT I * SW + 5
250  NEXT I
260  VTAB 21: PRINT  TAB( 8)"10 20 30 40 50 60 70 80 90 >90"
270  GOSUB 1000: REM  WARTEN
490  TEXT : HOME
500  PRINT "ENDE IN 500": END
600  DATA  10.3,16.4,14.6,13.4,14.2,11.9,8.9,7.8,2.4,0.2
1000  REM  UP WARTEN
1010  VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!"
1020  NORMAL
1030  GET A$: IF A$ = "" THEN 1030
1040  RETURN
```

Dieses Programm soll kurz erläutert werden:

Sätze	Inhalt
3Ø- 5Ø	Kommentar und Überschrift
6Ø	Abwarten
7Ø- 8Ø	Vorgabe der Klassenzahl (bei Bedarf muß N anders belegt werden); wenn N > 39, sollten Klassen zusammengelegt werden, damit ein Bildschirm zur Ausgabe ausreicht;
9Ø	Dimensionierung
1ØØ	Einlesen der Ausgangsdaten
11Ø-14Ø	Bestimmung der maximalen Häufigkeit
15Ø-17Ø	Alle auszugebenden Häufigkeiten werden so umgerechnet, daß der Maximalwert 35 Zeilen überdeckt.
18Ø	Vorgabe einer Schrittweite (bei mehr als 12 Klassen muß SW=2, bei mehr als 17 Klassen muß SW=1 sein; siehe auch oben 7Ø-8Ø)
2ØØ-21Ø	Vorbereiten des Zeichnens
22Ø	Waagrechte Linie am unteren Rand
23Ø-25Ø	Zeichnen senkrechter Histogrammbalken
26Ø	Ausgabe einer numerischen Fußzeile; diese ist bei anderen Problemstellungen entsprechend zu ändern.
27Ø	Abwarten
49Ø	Zurückschalten in Textmodus und Schlußausdruck
1ØØØ-1Ø4Ø	Unterprogramm zum Abwarten

4.2.8 Beispiel 5: Ballspiel

Aus den Graphikelementen, die in den vorangegangenen Beispielen verwendet wurden, läßt sich leicht - und dies soll das letzte Beispiel in diesem Zusammenhang sein - das "Grundmodell" eines sog. Bewegungsspiels entwickeln.

Es handelt sich bei dem folgenden Programm um die einfachste Version eines "Ballspiels", wobei der Ball durch ein sich bewegendes Rechteck dargestellt wird, welches zwischen den "Spielfeldumrandungen" (siehe 4.2.4) hin und her pendelt.

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 12)"4.7 - BALLSPIEL"
60  GOSUB 1000: REM  WARTEN
70  GR
80  VTAB 23: HTAB 6: PRINT "BITTE 12 SEKUNDEN WARTEN.   "
90  COLOR= 12: REM  GRUEN
100  REM  RAHMEN
110  VLIN 0,39 AT 0: VLIN 0,39 AT 39
120  HLIN 0,39 AT 0: HLIN 0,39 AT 39
130  REM  STARTPOSITION
140 X = 10:Y = 25
150  REM  SCHRITTWEITE UND NEUE KOORDINATEN
155 XV = 1:YV = 1
160 NX = X + XV:NY = Y + YV
170  IF NX > 38 THEN NX = 38:XV =  - XV
180  IF NX < 1 THEN NX = 1:XV =  - XV
190  IF NY > 38 THEN NY = 38:YV =  - YV
200  IF NY < 1 THEN NY = 1:YV =  - YV
210  COLOR= 12: PLOT NX,NY: REM  GRUENER BALL
220  COLOR= 0: PLOT X,Y: REM  LOESCHEN ALTE POSITION
230 X = NX:Y = NY
240 I = I + 1: IF I <  = 250 THEN 160
250  VTAB 24: INPUT "WEITER ? (J/N) ";A$
260  IF A$ = "J" THEN I = 0: GOTO 160
270  TEXT : HOME : PRINT "NA DENN NICHT": END
1000  REM  UP WARTEN
1010 VTAB 23: HTAB 6: INVERSE : PRINT "BITTE EINE TASTE DRUECKEN
!": NORMAL
1020  GET A$: IF A$ = "" THEN 1020
1030  RETURN
```

Sätze	Inhalt
3Ø- 6Ø	Vorbemerkungen und Abwarten.
7Ø	Umschalten auf Graphikmodus.
8Ø	Löschen der Fußzeile.
9Ø	Vorgabe der Rahmenfarbe.
1ØØ-12Ø	Umrandung des "Spielfeldes".
13Ø-14Ø	Vorgabe der Startposition (kann verändert werden).
15Ø-155	Vorgabe von Schrittweiten (sie bestimmen die Bewegungsgeschwindigkeit und die -richtung des "Balls").
16Ø	Bestimmung der neuen "Ball"-Koordinaten.
17Ø-2ØØ	Wird der "Spielfeldrand" erreicht, kehrt sich die Bewegungsrichtung des "Balls"um (hier könnten z.B. Rückprallwinkel und Rückprallgeschwindigkeit alternativ festgesetzt werden).
21Ø-22Ø	An der neuen Position wird ein grüner "Ball" gezeichnet, an der alten Position ein schwarzer "Ball", d.h. an der alten Position wird faktisch gelöscht (auf schwarzem Bildschirm), so daß der optische Eindruck einer Bewegung entstehen kann.
23Ø	Der zuletzt erreichte Punkt wird hinfort als der "alte" Punkt betrachtet, der im nächsten Durchlauf dieses Programmteils (16Ø-24Ø) zu löschen ist.
24Ø	Solange I nach der Erhöhung um 1 kleiner als 25Ø ist (dieser Wert ist willkürlich gewählt), wiederholt sich der Prozeß von 16Ø bis 24Ø, d.h. der "Ball" läuft 25Ø Schritte weit.

Sätze	Inhalt
25Ø-26Ø	Falls weiter gespielt werden soll, beginnt das Programm ab 16Ø mit 25Ø weiteren Schritten.
27Ø	Schlußausdruck.
1ØØØ-1Ø3Ø	Abwarten (Unterprogramm).

Würde man nach dem gleichen Muster innerhalb des gleichen "Spielfeldes" gleichzeitig einen zweiten "Ball" fliegen lassen (sinnvollerweise mit anderen Startkoordinaten und mit anderen Bewegungs- bzw. Schrittweiteparametern), so könnte man durch Koordinatenvergleich bei jedem Schleifendurchlauf (16Ø bis 24Ø) prüfen, ob die beiden "Bälle" sich berühren.

In einem solchen Berührungsfall könnte mit der Meldung

B U M M

das Programm abbrechen.

Der Leser versuche, selbst ein derartiges Programm zu erstellen.

4.3 Hochauflösende Graphik

Es wurde schon erwähnt, daß man bei der hochauflösenden Graphik sich den Bildschirm aufgeteilt denken muß in 192 Zeilen und 28Ø Spalten. Jeder der so entstehenden 53760 Punkte ist einzeln ansprechbar, so daß wir die Möglichkeit haben, sehr "feine" Linienzüge durch den APPLE zeichnen zu lassen. Insbesondere im mathematischen Bereich ist dies von großem Nutzen, wie die folgenden Beispiele zeigen.

Ergänzend ist anzumerken, daß auch hier - wie im Fall der Normalgraphik - am unteren Bildschirmrand Textzeilen per Voreinstellung freigehalten werden, so daß uns effektiv für die hochauflösende Graphik bei den 28Ø Spalten nur 16Ø (gedachte) Zeilen zur Verfügung stehen.
Bei der Verwendung der hochauflösenden Graphik ist allerdings auf einen besonderen Umstand zu achten:
Der Speicherbedarf ist beträchtlich, und es kann deshalb leicht passieren, daß zu lange Programme die Bildschirmausgabe der Graphik "stören" (und damit dann in der Regel zunichte machen).

Dieses Problem kann aber gelöst werden:

Wenn solche Störeffekte auftreten - sie sind in der Regel daran erkennbar, daß der Rechner anstelle einer schönen Graphik in unregelmäßiger Weise Pünktchen über den Bildschirm verstreut - so können diese mit den Anweisungen HIMEM und LOMEM gelöst werden.

Es ist dabei zu beachten, daß für die hochauflösende Graphik im APPLE-Rechner zwei Pufferbereiche zur Verfügung stehen, so daß alternativ zwei "Bildschirmseiten" ausgegeben werden können.

Soll der Pufferbereich I verwendet werden (wir gehen dabei von Rechnern mit mehr als 32 KByte aus), dann ist es sinnvoll, LOMEM bei der Adresse 16384 festzulegen.
Dann wird das Benutzerprogramm oberhalb dieser Adresse gespeichert und kann somit die Graphikausgabe nicht stören. (Nähere Einzelheiten sind den Benutzerhandbüchern zu entnehmen).

Wenn allerdings mit nicht zu großen Programmen gearbeitet wird, braucht man sich um die obigen Anmerkungen nicht zu kümmern.

4.3.1 "Einschalten" der hochauflösenden Graphik

Die hochauflösende Graphik wird vorbereitet mit der Anweisung

```
HGR
```

Sie löscht den ersten Graphik-Pufferbereich und stellt ihn hinfort für die hochauflösende Graphikausgabe zur Verfügung (entsprechend HGR2 für den zweiten Pufferbereich). Die Bildschirmaufteilung ist - wie schon erwähnt - 28Ø Spalten und 16Ø Zeilen, zuzüglich 4 Textzeilen am unteren Bildschirmrand.

Beim zweiten Pufferbereich stehen keine Textzeilen zur Verfügung, es sei denn, man gibt ein:

```
HGR2: POKE-163Ø1,Ø
```

(allerdings müssen dann auch die auszugebenden Textdaten per POKE gespeichert worden sein; dies ist recht umständlich, weshalb diese Möglichkeit, die ja nur in Ausnahmefällen wichtig wird, hier nicht weiter verfolgt werden soll).

Auch hier erfolgt die "Zurückschaltung" in den Textmodus mit der Anweisung

```
TEXT
```

4.3.2 Farben

Beim Zeichnen in hochauflösender Graphik kann man sich sechs verschiedener Farben bedienen (Farbkarte und Farbmonitor natürlich wieder vorausgesetzt).

Die entsprechende Anweisung lautet:

```
HCOLOR = Zahl
```

An der Stelle "Zahl" steht ein Wert zwischen Ø und 7 gemäß der folgenden Farbtabelle:

Farben:

Zahl	Farbe	Zahl	Farbe
Ø	schwarz	4	schwarz
1	grün	5	orange
2	violett	6	blau
3	weiß	7	weiß

4.3.3 Graphik-Anweisungen

Wie schon bei der Normalgraphik sind die wesentlichen Graphik-Anweisungen auch hier die zum Zeichnen eines Punktes bzw. zum Zeichnen einer geraden Linie.

Zum Zeichnen eines Punktes benutzen wir die Anweisung

```
HPLOT  S, Z
```

Diese Anweisung setzt einen Punkt an die Kreuzung der Spalte Nr. S mit der Zeile Nr. Z.

Die Anweisung

```
HPLOT  S1, Z1  TO  S2, Z2
```

zeichnet eine gerade Linie zwischen den Punkten P_1 (S_1, Z_1) und P_2 (S_2, Z_2).

Durch Erweiterung dieser Anweisung kann auch eine Folge von Linien gezeichnet werden, wie das folgende Beispiel verdeutlicht:

```
HPLOT  1Ø,1Ø TO 279,159 TO 5Ø,5Ø
```

4.3.4 Beispiel 1: Koordinatensystem

In einem ersten Beispiel für hochauflösende Graphik zeigen wir, wie ein rechtwinkliges Koordinatensystem an beliebiger Stelle des Bildschirms positioniert werden kann. Vorzugeben sind dafür die Spalte S und die Zeile Z des Koordinatenursprungs (S muß zwischen Ø und 279, Z zwischen Ø und 159 liegen).

ACHTUNG! :	Wie bei vielen anderen Programmen auch, ist hier nicht vorgesehen, fehlerhafte Eingaben "abzufangen" (der Leser kann solche "Sicherungen" leicht nachträglich in das Programm einbauen).

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT  TAB( 11)"4.8 - ACHSENKREUZ"
60  VTAB 20: PRINT "BITTE CONT EINGEBEN": STOP
70  HOME
80  INPUT "SPALTE DES URSPRUNGS  : ";S
90  PRINT : INPUT "ZEILE DES URSPRUNGS   : ";Z
100  HGR : HCOLOR= 7: REM  WEISS
110  HPLOT 0,Z TO 279,Z
120  HPLOT S,0 TO S,159
130  VTAB 23: PRINT "BITTE CONT EINGEBEN": STOP
140  TEXT : HOME : PRINT "ENDE": END
```

Es ist bei diesem und auch bei den folgenden Programmen bei Verwendung eines Schwarz-Weiß-Monitors darauf zu achten, welche "Farbe" in der Anweisung HCOLOR ausgewählt wird. Es kann bei bestimmten Farben und/oder bei bestimmten Richtungen der dünnen, hochaufgelösten Linien geschehen, daß die Bildschirmausgabe dieser Linien unterdrückt wird.
Durch Ausprobieren verschiedener "Farben" kann dieses Problem gelöst werden.

4.3.5 Beispiel 2: Lineare Funktion

Soll in ein Achsenkreuz eine Gerade eingezeichnet werden, so beschränkt man sich in erster Linie auf den ersten Quadranten, legt also den Achsenursprung nach links unten - so auch auf dem Bildschirm.

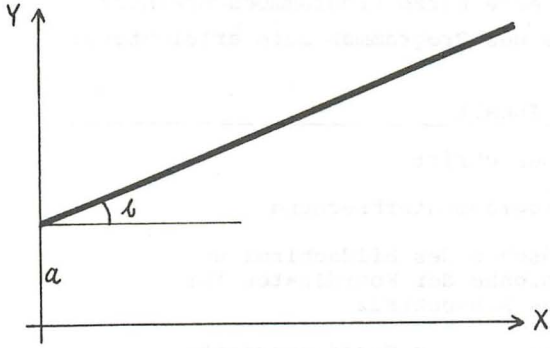
Dieser Punkt ist auf dem hochauflösenden Bildschirm gekennzeichnet durch die Spalte S=Ø und die Zeile Z=159. Mit diesen Vorgaben kann hier also auf das Programm des vorangegangenen Abschnitts zurückgegriffen werden.

Die Lage der Geraden im Achsenkreuz wird durch die beiden Parameter

- Ordinatenabschnitt a
- Steigung (Tangens des Steigungswinkels) b

bestimmt, gemäß der Funktionsgleichung:

$$y_i = a + bx_i$$



Das Programm muß also die beiden Geradenparameter als Inputinformationen bekommen. Dann können ohne Schwierigkeiten Achsenkreuz und Gerade gezeichnet werden.

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 13)"4.9 - GERADE"
60  VTAB 20: PRINT "BITTE CONT EINGEBEN": STOP
70  HOME :S = 0:Z = 159
80  INPUT "ORDINATENABSCHNITT A : ";A
90  PRINT : INPUT "GERADENSTEIGUNG B     : ";B
100  HGR : HCOLOR= 7: REM  WEISS
110  HPLOT 0,Z TO 279,Z
120  HPLOT S,0 TO S,159
130  FOR X = 0 TO 279
140 Y = 159 - A - B * X
150  IF Y < 0 OR Y > 159 THEN 170
160  HPLOT X,Y
170  NEXT X
180  VTAB 23: PRINT "ENDE DER AUSGABE": END
```

Auch hier soll eine kurze Programmbeschreibung das Verständnis des Programmablaufs erleichtern:

Sätze	Inhalt
3Ø- 5Ø	Überschrift
6Ø	Programmunterbrechung
7Ø	Löschen des Bildschirms und Vorgabe der Koordinaten für das Achsenkreuz
8Ø- 9Ø	Vorgabe der Geradenparameter
1ØØ	Umschalten auf hochauflösende Graphik
11Ø-12Ø	Zeichnen des Achsenkreuzes
13Ø-17Ø	Zeichnen der Geraden 13Ø Vorgabe der X-Koordinate; 14Ø Bestimmung der Y-Koordinate unter Beachtung der Zeileneinteilung des Bildschirms 15Ø Abfangen unzulässiger Y-Werte 16Ø Zeichnen eines Punktes der Geraden 17Ø Nächster Punkt
18Ø	Schlußausdruck und Beendigung des Programms.

4.3.6 Beispiel 3: Kreis

Daß mit der hochauflösenden Graphik auch gekrümmte Linien gezeichnet werden können, zeigt am einfachsten das Beispiel des Kreises.

Inputinformationen müssen jetzt die Koordinaten des Mittelpunkts und der Radius des Kreises sein.

Da generell das Zeichnen so vonstatten gehen kann, daß Punkt für Punkt gezeichnet wird, muß also zu jedem im Funktionsbereich liegenden X_i-Wert (= Spaltenkoordinate) der über die mathematische Beziehung feststellbare Y_i-Wert (= Zeilenkoordinate) bestimmt werden.
Dazu benötigt man die Kreisgleichung, die - bei beliebigem Mittelpunkt - folgendermaßen lautet:

$$(X_i - X_o)^2 + (Y_i - Y_o)^2 = r^2$$

wobei $(Y_{1/2})_i$ = Y-Koordinaten

X_i = X-Koordinaten

r = Radius

X_o = X-Koordinate des Mittelpunkts

Y_o = Y-Koordinate des Mittelpunkts

Die X-Koordinatenwerte bewegen sich dabei natürlich im Bereich zwischen $r-X_o$ und $r+X_o$, die Werte $(Y_1)_i$ konstituieren den oberen Halbkreis, die Werte $(Y_2)_i$ den unteren.

Es ist dabei in diesem Zusammenhang daran zu erinnern, daß auf dem Bildschirm die oberen Zeilen niedrigere Z-Werte aufweisen (= Y-Koordinaten) als die unteren Zeilen. Weiterhin ist dafür Sorge zu tragen, daß bei Variation der X- und der Y-Werte die zulässigen Grenzen nicht überschritten werden.

Ein entsprechendes Programm sieht dann folgendermaßen aus:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 13)"4.10 - KREIS"
60  VTAB 20: PRINT "BITTE CONT EINGEBEN": STOP
70  HOME :S = 0:Z = 159
80  INPUT "X-KOORDINATE DES MITTELPUNKTES : ";A
90  PRINT : INPUT "Y-KOORDINATE DES MITTELPUNKTES : ";B
95  PRINT : PRINT : PRINT : INPUT "RADIUS   : ";R
100  HGR : HCOLOR= 7: REM  WEISS
110  HPLOT 0,Z TO 279,Z
120  HPLOT S,0 TO S,159
130  FOR X = A - R TO A + R
140 H = R * R - (X - A) ^ 2
150  IF H <  = 0 THEN 200
160 Y1 = B +  SQR (H):Y2 = B -  SQR (H)
170  IF X < 0 OR X > 279 OR Y1 < 0 OR Y1 > 159 THEN 190
180  HPLOT X,Y1
190  IF X < 0 OR X > 279 OR Y2 < 0 OR Y2 > 159 THEN 200
195  HPLOT X,Y2
200  NEXT X
210  VTAB 23: PRINT "ENDE DER AUSGABE": END
```

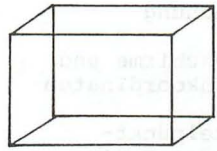
Auch hier fügen wir eine Programmbeschreibung an, bei der wir uns aber jetzt etwas kürzer fassen können, weil viele Programmbestandteile auch im vorangegangenen Beispiel schon auftauchten:

Sätze	Inhalt
3Ø- 5Ø	Überschrift
6Ø	Programmunterbrechung
7Ø	Löschen des Bildschirms und Angabe der Achsenkoordinaten
8Ø- 95	Vorgabe der Mittelpunkt-Koordinaten und des Radius
1ØØ	Umschalten auf hochauflösende Graphik
11Ø-12Ø	Zeichnen des Achsenkreuzes
13Ø-2ØØ	Zeichnen des Kreises
	13Ø Angabe des Wertebereichs der X-Koordinaten
	14Ø Bestimmung einer Hilfsgröße H
	15Ø Abfangen unzulässiger H-Werte (aus H wird die Wurzel gezogen!)
	16Ø Bestimmung der Y-Koordinaten für den unteren und den oberen Halbkreis
	17Ø Abfangen unzulässiger Koordinaten
	18Ø Zeichnen des unteren Halbkreises
	19Ø Abfangen unzulässiger Koordinaten
	195 Zeichnen des oberen Halbkreises
	2ØØ Nächster Punkt
21Ø	Ende des Programms

4.3.7 Beispiel 4: Dreidimensionale Graphik

Natürlich ist es auch möglich, dreidimensionale Graphiken zu erzeugen - wenn auch etwas aufwendiger.

Dies zeigt das folgende Beispiel eines Quaders:



Das entsprechende Programm, das der Einfachheit halber mit festen Vorgaben für Länge, Breite, Tiefe und Betrachtungswinkel arbeitet (selbstverständlich könnten aber diese Informationen auch über INPUT alternativ vom Benutzer vorgegeben werden), stellt sich folgendermaßen dar:

```
30  PRINT  TAB( 9)"APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : PRINT  TAB( 9)"R.PRUST / W.VOSS 1984"
50  PRINT : PRINT : PRINT  TAB( 12)"4.11 - QUADER"
60  VTAB 20: PRINT "BITTE CONT EINGEBEN": STOP
70  HOME
80 X1 = 50:X2 = 100:X3 = 70:X4 = 120
90 Y1 = 50:Y2 = 80:Y3 = 30:Y4 = 60
100  HGR : HCOLOR= 7: REM  WEISS
110  HPLOT X1,Y1 TO X2,Y1 TO X2,Y2 TO X1,Y2 TO X1,Y1
130  HPLOT X3,Y3 TO X4,Y3 TO X4,Y4 TO X3,Y4 TO X3,Y3
140  HPLOT X1,Y1 TO X3,Y3
150  HPLOT X2,Y1 TO X4,Y3
160  HPLOT X2,Y2 TO X4,Y4
165  HPLOT X1,Y2 TO X3,Y4
170  VTAB 23: PRINT "ENDE": END
```

Man könnte ein derartiges Programm nun so weiterführen, daß die Eckpunkte des Quaders einer gleichmäßigen, z.B. kreisförmigen Bewegung unterworfen werden. Man erhält dann sich bewegende dreidimensionale Bilder, die andeuten, wie derartige Konzepte im Konstruktionswesen oder im Architekturwesen verwendet werden können.

Für derartige Probleme gibt es aber eine Reihe geeigneter Hilfsprogramme und Software-Angebote, so daß die Mühe eigener Programmierung hier eher unangebracht ist.

4.4 Das Mischen von Text und Graphik

Häufig ist es sinnvoll, daß auf einer Bildschirmausgabe Graphik und Text gemischt werden.

Wünschenswert ist etwa die Beschriftung der Achsen des Achsenkreuzes, die Kennzeichnung von Reihen, zumindest wenn mehrere gleichzeitig in einer Graphik auftauchen u.ä.

Es bieten sich dafür verschiedene Möglichkeiten an, die aber alle nicht voll befriedigen können:

1. Bestimmte, erläuternde Texte können in die unteren Zeilen des Graphikbildschirms eingesetzt werden, wie dies in vorangegangenen Beispielen schon geschehen ist.

2. In eine Graphik einzusetzende Schriftzeichen oder Ziffern könnten auch "eingezeichnet" werden, indem diese Symbole gewissermaßen als Graphikbestandteile behandelt werden.

3. Geeignete Hilfsprogramme stehen zur Verfügung, die auf geschickte Weise das Mischen von Graphik und Text erlauben.

Zu Punkt 1. braucht hier nichts mehr gesagt zu werden; wohl aber zu den beiden folgenden Punkten.

4.4.1 "Zeichnen" von Symbolen

Betrachtet man sich einmal ein Symbol auf dem Bildschirm genauer, z.B. die 1, so sieht man, daß diese Ziffer aus einer Reihe von Punkten aufgebaut ist. Diese Punkte könnte man natürlich mit einer Reihe von HPLOT-Anweisungen in einen auf den Graphikmodus umgeschalteten Bildschirm hineinzeichnen.

Diese Vorgehensweise, die zunächst umständlich scheint, bietet drei Vorteile:

1. Symbole erscheinen auf dem Graphikbildschirm (das ist ja auch das Ziel dieser Prozedur).

2. Diese Symbole können an beliebige Bildschirmstellen plaziert werden.

3. Die Symbole können in unterschiedlicher Größe ausgegeben werden.

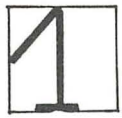
Das folgende Programm ist ein derartiger "Grundbaustein":

```
30  PRINT  TAB( 8)"APPLE - TIPS UND TRICKS"
40  PRINT : PRINT : PRINT
50  PRINT  TAB( 8)"R.PRUST / W.VOSS,  1984"
60  PRINT : PRINT : PRINT : PRINT  TAB( 9)"4.13 - SYMBOLZEICHNEN":
PRINT : PRINT
70  PRINT : PRINT : PRINT "DIESES PROGRAMM ZEICHNET DAS SYMBOL
 1"
80  PRINT "IN UNTERSCHIEDLICHER GROESSE, ALTERNA-"
90  PRINT "TIV WAEHLBARER STRICHSTAERKE UND AN BE-"
95  PRINT "LIEBIGE POSITIONEN DES BILDSCHIRMS"
96  PRINT : PRINT "ACHTUNG : ": PRINT : PRINT "DIESES PROGRAMM
PRUEFT NICHT AUF UNZU-": PRINT "LAESSIGE WERTE BEI DER GRAPHIK
!!!"
97  GOSUB 5000: REM  WARTEN
120  GOSUB 1000: REM  1
130  PRINT : PRINT "ENDE": END
1000  REM  UP 1
1010  HOME : INPUT "VERGROESSERUNGSFAKTOR : ";F%
1015  PRINT : INPUT "STRICHSTAERKE          : ";ST
1020  PRINT : INPUT "START (S,Z) : ";S,Z
1025  HGR : HCOLOR= 3
1040  FOR K = 1 TO ST
1050  HPLOT S + F% * 2,Z TO S + F% * 3,Z
1070  HPLOT S + F% * 3,Z TO S + F% * 3,Z + F% * 8
1080  HPLOT S + F%,Z + F% * 8 TO S + F% * 5,Z + F% * 8
1090  HPLOT S,Z + F% * 2 TO S + F% * 2,Z
1110 S = S + 1:Z = Z + 1
1120  NEXT K
1130  GOSUB 5000: REM  WARTEN
1140  TEXT : RETURN
5000  REM  UP WARTEN
5010  VTAB 23: CALL  - 868: INVERSE : HTAB 6: PRINT "BITTE EINE
TASTE DRUECKEN !": NORMAL : POKE  - 19368,0: WAIT  - 16384,128:
POKE  - 16368,0: VTAB 23: CALL  - 868: RETURN
```

Dieses Programm zeichnet in ein vorgegebenes Feld die Ziffer 1, wobei Größe und Strichstärke alternativ vorgegeben werden können. Die Lage des zu belegenden Feldes wird fixiert durch Spalte und Zeile der Ecke oben links.

So wie das Problem hier für die Ziffer 1 gelöst wurde, so könnten auch Programme für die anderen Ziffern, für Buchstaben und für Sonderzeichen entwickelt, als "Bausteine" in Reserve gehalten und bei Bedarf verwendet werden.

Eine kurze Programmbeschreibung soll die Funktionsweise erläutern:

Sätze	Inhalt
3Ø- 96	Überschrift und Erläuterungen zum Programm
97	Sprung ins UP 5ØØØ: Warten
12Ø	Sprung ins UP 1ØØØ zum Zeichnen
13Ø	Beendigung des Programms
1Ø1Ø-1Ø15	Eingabe von Vergrößerungsfaktor und Strichstärke der zu zeichnenden 1.
1Ø2Ø	Vorgabe der Startkoordinaten auf dem Bildschirm (S=Spalte, Z=Zeile). (Es handelt sich dabei um die linke obere Ecke des Feldes, in das die 1 gezeichnet wird, wie nebenstehende Skizze verdeutlicht). 
1Ø25	Umschalten auf Graphikmodus und Farbvorgabe.
1Ø4Ø-112Ø	Zeichnen des Symbols 1 1Ø4Ø: Wiederholung so oft, wie die Strichstärke ST vorgibt 1Ø5Ø-1Ø9Ø: Zeichnen 111Ø: Verschiebung gemäß Strichstärke

Sätze	Inhalt
113Ø	Sprung ins UP 5ØØØ: Warten
114Ø	Umschalten in Textmodus und Rücksprung
5ØØØ	Beginn des UP Warten
5Ø1Ø	Ausgabe einer Meldung in inverser Schrift; Abwarten einer Tasteneingabe und Rücksprung

Man erkennt aus dieser Beschreibung, daß mit jeweiliger Zeilen- und Spaltenverschiebung um 1 das Symbol in 1Ø5Ø-1Ø9Ø so oft wiederholend gezeichnet wird, wie durch die Strichstärkenangabe verlangt wird.

Der "Kern" des Programms sind also die Sätze 1Ø5Ø bis 1Ø9Ø: Sollte ein anderes Symbol als die 1 gezeichnet werden, müßten an dieser Stelle entsprechend andere HPLOT-Anweisungen stehen.

4.4.2 Hilfsprogramme zum Mischen von Text und Graphik

Das Mischen von Text und Graphik ist bei den meisten Rechnern nicht ganz unproblematisch, insbesondere wenn es um die hochauflösende Graphik geht.
Wenn nämlich auf diesen Graphikmodus umgeschaltet wird, dann stehen zwar auf der ersten von zwei Bildschirmseiten die unteren beiden Zeilen noch für Texte zur Verfügung, bei der zweiten Bildschirmseite aber nichts mehr.

Und auch die genannten unteren zwei Zeilen nützen nicht allzuviel, wenn man beispielsweise Buchstaben und/oder Ziffern "mitten" in eine Graphik hineinplazieren möchte - etwa zur Beschriftung von Achsen eines Koordinatensystems oder zur Kennzeichnung von Zeitreihen oder von mathematischen Funktionen, die gezeichnet wurden, und dergl.

Dieses Problem kann auch mit geeigneten Graphik-Hilfsprogrammen gelöst werden, über die in diesem Buch aber nicht gesprochen werden soll. Der interessierte Leser wird entsprechende Informationen von jedem guten Software-Anbieter einholen können.

Abschließend sei darauf hingewiesen, daß man spezielle Symbole auch so erzeugen kann, daß man die betreffende Linienfolge in einer Vektorentabelle erfaßt, deren Binärcode ermittelt und diese Tabelle dann im Bedarfsfall aufruft.

Diese Methode ist recht umständlich und soll hier nicht näher erläutert werden, weil sich eine anschauliche Beschreibung bei Lon Poole, Martin McNiff und Steven Cook findet (APPLE II Anwendungshanbuch, Münche- 1981, Seite 6-13 ff.).

Kapitel 5: Verbesserung der Programmstrukturen

5.1 Vorbemerkungen

Unter dieser Überschrift sollen einige Befehle, die den Programmablauf betreffen, nach kurzer Einführung anhand von Programmbeispielen näher erläutert werden. Die Sprungbefehle GOTO und IF...THEN sind nicht die einzigen Anweisungen, die Sprünge innerhalb eines Programms ermöglichen. Der Befehl GOSUB wurde ebenfalls schon mehrfach verwendet, doch fehlten grundlegende Bemerkungen zu den Einsatzmöglichkeiten. Dies soll nun nachgeholt werden. Dabei wird deutlich werden, daß der Benutzer - wie in vielen anderen Fällen - auch bei der Gestaltung des Programmablaufs einen Kompromiß eingehen muß. Die beiden Optimalitätskriterien "schnellstmögliche Ausführung" und "geringster Speicherplatzbedarf" sind in der Regel nicht gleichzeitig zu verwirklichen. Kommen weitere Punkte hinzu - etwa übersichtliche Bildschirmgestaltung, Benutzerfreundlichkeit, gute "Lesbarkeit" des Programms, problemlose Veränderungsmöglichkeiten - so wird ein Kompromiß schwierig. Jeder einzelne Benutzer wird dann für sich festlegen, welcher am stärksten gewichtet wird von den vorher genannten Aspekten und wird eventuell auch noch weitere Punkte in seine Überlegungen mit einbeziehen.

Schließlich wird auch die spezielle Aufgabenstellung eine Rolle spielen und natürlich der potentielle Nutzerkreis. Wer nur für sich selbst programmiert, braucht sicher nicht so sehr auf Benutzerfreundlichkeit zu achten wie jemand, der Programme schreibt für Freunde und Bekannte, die vielleicht nicht so geübt sind in der Rechnerbedienung.
So wird sich denn jeder Leser aus den folgenden Anregungen vielleicht etwas anderes herausgreifen wollen für seine eigene Arbeit.

5.2 Unterprogrammtechnik

Auch ohne den Einsatz von Unterprogrammtechnik kann man programmieren und vermutlich gar nicht einmal so sehr schlecht. Der Verzicht auf Unterprogramme - und damit auf Sprünge - erleichtert es in jedem Fall dem Leser eines Programmlistings, die Arbeitsweise und den Ablauf eines Programms zu verstehen.
Doch hat dieser Programmierstil auch Nachteile. Wenn dieselben Anweisungsfolgen mehrfach ins Programm aufgenommen werden, wird das Programm u.U. wesentlich länger und dadurch wieder etwas weniger überschaubar, und natürlich wird mehr Speicherplatz benötigt.
Auch das Eintippen solcher Programme dauert länger als nötig wäre.
So wird denn auf Dauer niemand ganz auf Unterprogramme verzichten.

Unterprogramme sind Anweisungsfolgen, die meist für sich genommen vollständige Programme sind, möglicherweise ohne die erforderlichen Dateneingabebefehle; die Wertzuweisung für die Variablen erfolgt oft im sogenannten Hauptprogramm. Zumindest aber sind es Befehlsblöcke, die in der Regel eine klar abgegrenzte Aufgabe erfüllen. Als Beispiel könnten einige Programme des Kapitels 4 herangezogen werden: Im Hauptprogramm werden die für die Graphik erforderlichen Werte eingegeben bzw. berechnet, das Unterprogramm leistet das Zeichnen.
Es können "beliebig" viele Unterprogramme aufgenommen werden (die einzige Beschränkung stellt der Speicherplatz dar); Unterprogramme können ihrerseits Unterprogramme aufrufen.

Wenn eine Folge von Befehlen in einem Programm mehrfach auftaucht, ist es sinnvoll, sie als Unterprogramm zu gestalten, d.h. als eigenständigen Block, der am Anfang oder Ende des Hauptprogramms plaziert wird, und diesen Block mit dem Befehl GOSUB anzuspringen.
Der letzte Befehl des Unterprogramms muß die Anweisung RETURN sein. Dadurch erfolgt die Rückkehr ins Hauptprogramm zur ersten Anweisung hinter dem aufrufenden GOSUB.

Sehr kurze, häufig aufgerufene Unterprogramme - wie z.B. die bedingte Warteschleife durch GET oder WAIT - sollten bei langen Programmen an den Anfang gesetzt werden, längere Unterprogramme hingegen hinter den END-Befehl des Hauptprogramms. Der Grund liegt in der Ausführungsgeschwindigkeit. Jeder Sprung innerhalb eines Programms - ganz gleich, welche Sprunganweisung verwendet wird - kostet Zeit, und zwar abhängig von der Sprungweite. Bei jedem Sprung beginnt der Rechner das Programm von Anfang an zu durchsuchen nach der Zeile mit der angegebenen Sprungadresse. Unterprogramme mit niedrigen Zeilennummern werden also wesentlich schneller gefunden als solche mit hohen Zeilennummern.
Damit ein vernünftiges Verhältnis zwischen der für den Sprung benötigten Zeit und der Ausführungszeit des Unterprogramms erreicht wird, sollte die obige Regel beachtet werden. Allerdings müssen natürlich am Anfang stehende Unterprogramme durch GOTO übersprungen werden, damit sie nicht schon bei Start des Programms ausgeführt werden, so daß der Aufbau schematisch so dargestellt werden kann:

1. GOTO - Sprung zum Beginn des Programms.

2. Kurze Unterprogramme, die vom Hauptprogramm oder anderen Unterprogrammen aufgerufen werden.

3. Beginn des Hauptprogramms.
   "Initialisierung" durch Dimensionierung, Definition von Funktionen, Festlegung von Konstanten, eventuell Einlesen der Daten.
   Programmkern

4. END - Ende des Hauptprogramms.

5. Lange Unterprogramme.

Ein Nachteil des Unterprogrammeinsatzes auf dem APPLE - neben den schon erwähnten - liegt im Aufbau der Sprache BASIC begründet. Variablenwerte, die das Unterprogramm aus dem Hauptprogramm übernimmt, müssen im Hauptprogramm genau den Variablennamen zugewiesen sein, die auch das Unterprogramm verwendet. Bei mancher anderen Programmiersprache ist das komfortabler gestaltet.
Es können etwa zwei Variablenlisten einander gegenübergestellt werden, eine für die Variablen des Hauptprogramms, eine für die des Unterprogramms; über die Position in den Listen erfolgt die Übergabe der erforderlichen Werte.

Nun aber zu einem Anwendungsbeispiel: Wir kommen noch einmal auf das Sortierprogramm aus Kapitel 1 zurück, jetzt allerdings in etwas abgewandelter Form.
Es werden zwei Stringarrays eingelesen (N$ für Namen und A$ für Altersangaben).

Das Programm soll nacheinander erst das Namensarray sortieren, dabei das Altersarray mitführen und eine entsprechende Bildschirmausgabe machen und dann das Altersarray, wobei das Namensarray mitgeführt wird. Es erfolgt wiederum eine Bildschirmausgabe.
Die Ausgabe soll eine Überschrift umfassen und jeweils 5 Elemente aus den sortierten Arrays. Während des Sortiervorgangs erscheint ein Hinweis darauf, ob es sich um den ersten oder den zweiten Sortiervorgang handelt.
Das Programm wird in drei Versionen vorgestellt: eine ohne jedes Unterprogramm, eine mit geradezu ekzessiver Nutzung der Unterprogrammtechnik und eine Zwischenstufe.

Wir beginnen mit dem besonders abschreckenden Beispiel. Das Hauptprogramm leistet nichts anderes als die "Initialisierung", das bedeutet hier Dimensionierung, Festlegung von Konstanten und Einlesen der Daten, und die Variablenübergabe vom Hauptprogramm an die Unterprogramme, bzw. von den Unterprogrammen ans Hauptprogramm und als letztes den Ausdruck "E N D E".
Alles andere steht in den Unterprogrammen. Der Leser wird feststellen, daß das Programm sehr schwer lesbar ist, der Ablauf läßt sich kaum verfolgen. Am Bildschirm schon gar nicht. Außerdem ist der Nutzen im Beispiel relativ gering; dies liegt allerdings teilweise auch an der Wahl der Aufgabenstellung.
Der benötigte Speicherplatz ist größer als bei den beiden anderen Versionen, wie später in einer Übersicht deutlich wird.

NICHT ZU EMPFEHLEN

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM   ****  DEMONSTRATIONSBEISPIEL ***
11  REM    ** FUER UNTERPROGRAMMTECHNIK **
20  GOTO 200
30  VTAB 22: CALL  - 868: HTAB 6: INVERSE : PRINT "DRUECKEN
SIE IRGENDEINE TASTE": NORMAL : POKE  - 16368,0: WAIT
- 16384,128: POKE  - 16368,0: VTAB 22: CALL  - 868: RETURN

40 H$ = Y$(I):Y$(I) = Y$(J):Y$(J) = H$:H$ = X$(I):X$(I)
= X$(J):X$(J) = H$: RETURN
50  FOR Q = 1 TO 40: PRINT "*";: NEXT : PRINT : RETURN

60  HOME : GOSUB 50: HTAB 6: PRINT "S O R T I E R P R O
G R A M M": PRINT : GOSUB 50: RETURN
70 Z = 0: FOR I = 1 TO N: PRINT N$(I),A$(I):Z = Z + 1:
IF Z = M AND I < N THEN  GOSUB 90:Z = 0
80  NEXT I: GOSUB 90: RETURN
90  PRINT : GOSUB 50: GOSUB 30: GOSUB 60: RETURN
100  FOR I = 1 TO N - 1: FOR J = I + 1 TO N: IF Y$(I) >
Y$(J) THEN  GOSUB 40
110  NEXT J,I: RETURN
120  GOSUB 60: VTAB 12: HTAB 15: INVERSE : PRINT R;". RUNDE":
NORMAL : FOR Q = 1 TO 10,00: NEXT : RETURN
200  REM  BEGINN DES PROGRAMMS
210 N = 20
220  HOME : TEXT : DIM N$(N),A$(N),Y$(N),X$(N)
230 M = 5
240  FOR I = 1 TO N: READ N$(I),A$(I): NEXT
250 R = 1: GOSUB 120: FOR I = 1 TO N:Y$(I) = N$(I):X$(I)
= A$(I): NEXT : GOSUB 100
260  FOR I = 1 TO N:N$(I) = Y$(I):A$(I) = X$(I): NEXT :
GOSUB 60: GOSUB 70
270 R = 2: GOSUB 120: FOR I = 1 TO N:Y$(I) = A$(I):X$(I)
= N$(I): NEXT : GOSUB 100
280  FOR I = 1 TO N:A$(I) = Y$(I):N$(I) = X$(I): NEXT :
GOSUB 60: GOSUB 70
290  VTAB 12: HTAB 15: INVERSE : PRINT "E N D E": NORMAL
: END
1000  DATA     MUELLER,40,FRIEDRICHS,23,MEIER,17,MAIER,52
1010  DATA   KOCH,45,SCHUHMANN,34,NEUHAUS,21,MIES,57
1020  DATA  LUDWIG,14,HAUSMANN,33,ADAM,41,GRIMM,28
1030  DATA  OBERMANN,62,GROTE,24,NAGEL,58,MUTH,46
1040  DATA  FROEHLICH,11,SCHMITT,43,KLEIN,65,MARSCH,19
```

Wir wollen versuchen, durch eine Programmbeschreibung den Ablauf zumindest etwas deutlich werden zu lassen; dazu bedarf es allerdings einer ausführlichen Beschreibung:

Sätze	Inhalt
1Ø-11	Bemerkungen
2Ø	Sprung zum Beginn des Programms
3Ø	Unterprogramm fürs Unterbrechen
4Ø	Unterprogramm zum Vertauschen, falls in einem der Arrays eine falsche Reihenfolge festgestellt wurde.
5Ø	Unterprogramm für die Erzeugung einer gesamten Bildschirmzeile aus Sternchen. Wird im folgenden als Strich bezeichnet.
6Ø	Unterprogramm für die Überschrift des Bildschirms, benötigt den Strich aus Zeile 5Ø als Rahmen.
7Ø-9Ø	Unterprogramm für den Ausdruck. Die Anzahl der ausgegebenen Zeilen wird durch Z gezählt, nach jeweils 5 Zeilen (M=5) und nach der letzten Zeile wird das Unterprogramm in Zeile 9Ø angesprungen. Dies leistet die Darstellung des Strichs, das Warten und das Neuanzeigen der Überschrift durch Sprung zu den jeweiligen Unterprogrammen.
1ØØ-11Ø	Unterprogramm für den Vergleich des zu sortierenden Arrays Y$.
12Ø	Unterprogramm für den Ausdruck erste bzw. zweite Runde. Es benötigt Unterprogramm 6Ø für die Neuerzeugung der Überschrift.

Sätze	Inhalt
2ØØ	Beginn des Hauptprogramms
21Ø-23Ø	"Initialisierung" - es müssen 4 Arrays dimensioniert werden, damit die Subroutinen 4Ø und 1ØØ in dieser allgemeinen Form verwendet werden können.
24Ø	Einlesen der Daten
25Ø	Der Wert 1 für R muß an das Unterprogramm 12Ø übergeben werden, das die Meldung über die Runde ausgibt. Danach müssen die Arrays N$ und A$ an die Subroutinenarrays Y$ und X$ übergeben werden. Y$ wird sortiert, X$ mitgeführt. Uprog 1ØØ leistet die Vergleiche innerhalb des Y$-Arrays und ruft seinerseits bei Bedarf Uprog 4Ø zum Vertauschen auf.
26Ø	Die umgeordneten Listen müssen an die Ursprungsarrays übergeben werden. Danach wird wieder Uprog 6Ø für die Überschrift aufgerufen und dann Uprog 7Ø für den Ausdruck der Liste (das seinerseits Uprog 9Ø aufruft, das wiederum Uprog 5Ø, Uprog 3Ø und Uprog 6Ø aufruft, welches ebenfalls Uprog 5Ø aufruft - alles klar?)
27Ø	Ähnlich wie in 25Ø müssen die Werte für R an das Unterprogramm 12Ø und die Arraywerte A$ und N$ an das Unterprogramm 1ØØ übergeben werden, das für das Sortieren (mit Hilfe von Uprog 4Ø) zuständig ist.
28Ø	Entspricht Zeile 26Ø: Die sortierten Werte werden wieder ans Hauptprogramm übergeben und es folgt der Ausdruck.

Sätze	Inhalt
29Ø	Mit einem entsprechenden Bildschirmhinweis endet das Programm
1ØØØ-1Ø4Ø	Spieldaten in DATA-Zeilen

Wenn ein Programm so aufgebaut ist, ist es ohne Programmbeschreibung recht schwierig, die Arbeitsweise zu verstehen. Spätestens auf der vierten GOSUB-Ebene weiß man nur noch selten, von welcher Zeile aus der Aufruf erfolgte, wenn man das Programm nicht gerade selbst geschrieben hat, und zwar am Tag vorher.
Zwar ist die Wiederholung von Anweisungsfolgen durh die große Anzahl von Unterprogrammen weitgehend vermieden, aber es wurden auch zusätzliche Befehle notwendig, nämlich für die Übergabe von Variablenwerten. Außerdem erforderte diese Art der Programmierung die Dimensionierung von 2 zusätzlichen Arrays. Falls N nicht gerade nur 2Ø beträgt, macht sich diese "Verschwendung" sehr stark am Platzbedarf bemerkbar.

Die zweite Version benutzt die Unterprogramme 3Ø (Warten), 4Ø (Vertauschen), 5Ø (Strich) und 7Ø (Ausdruck). Alle anderen wurden gestrichen, die entsprechenden Anweisungen verlagert. Da auch 9Ø gestrichen wurde, mußte die Ausdrucksubroutine etwas abgeändert werden. Sie ist jetzt die Einzige, die andere Unterprogramme aufruft.

Durch Wahl der Zeilennummern wurde versucht, die Parallelen zur vorigen Version aufzuzeigen und die Unterschiede deutlich werden zu lassen.
Das Unterprogramm 6Ø für die Überschrift konnte durch die wesentliche bessere Lösung in Zeile 245 (mit Hilfe des Bildschirmfensters) ersetzt werden:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM   ***** DEMONSTRATIONSBEISPIEL ***
11  REM    ** FUER UNTERPROGRAMMTECHNIK **
20  GOTO 200
30  VTAB 22: CALL  - 868: HTAB 6: INVERSE : PRINT "DRUECKEN
SIE IRGENDEINE TASTE": NORMAL : POKE  - 16368,0: WAIT
- 16384,128: POKE  - 16368,0: HOME : RETURN
40 H$ = N$(I):N$(I) = N$(J):N$(J) = H$:H$ = A$(I):A$(I)
= A$(J):A$(J) = H$: RETURN
50  FOR Q = 1 TO 40: PRINT "*";: NEXT : PRINT : RETURN

70  HOME :Z = 0: FOR I = 1 TO N: PRINT N$(I),A$(I):Z =
Z + 1: IF Z = M OR I = N THEN Z = 0: PRINT : GOSUB 50:
GOSUB 30: REM     STRICH UND WARTEN
80  NEXT I: RETURN
200  REM  BEGINN DES PROGRAMMS
210 N = 20
220  TEXT : HOME : DIM N$(N),A$(N)
230 M = 5
240  FOR I = 1 TO N: READ N$(I),A$(I): NEXT
245  GOSUB 50: HTAB 6: PRINT "S O R T I E R P R O G R A
M M": PRINT : GOSUB 50: POKE 34,6: HOME
250  VTAB 12: HTAB 17: INVERSE : PRINT "1. RUNDE": NORMAL

253  FOR I = 1 TO N - 1: FOR J = I + 1 TO N: IF N$(I) >
N$(J) THEN  GOSUB 40: REM  VERTAUSCHUNG
256  NEXT J,I
260  GOSUB 70: REM    AUSDRUCK DER SORTIERTEN LISTE
270  VTAB 12: HTAB 17: INVERSE : PRINT "2. RUNDE": NORMAL

273  FOR I = 1 TO N - 1: FOR J = I + 1 TO N: IF A$(I) >
A$(J) THEN  GOSUB 40: REM  VERTAUSCHUNG
276  NEXT J,I
280  GOSUB 70: REM   AUSDRUCK DER SORTIERTEN LISTE
290  VTAB 12: HTAB 15: INVERSE : PRINT "E N D E": NORMAL
: POKE 34,0: PRINT : END
1000  DATA     MUELLER,40,FRIEDRICHS,23,MEIER,17,MAIER,52
1010  DATA   KOCH,45,SCHUHMANN,34,NEUHAUS,21,MIES,57
1020  DATA  LUDWIG,14,HAUSMANN,33,ADAM,41,GRIMM,28
1030  DATA  OBERMANN,62,GROTE,24,NAGEL,58,MUTH,46
1040  DATA  FROEHLICH,11,SCHMITT,43,KLEIN,65,MARSCH,19
```

Wir meinen, daß hier auf eine Programmbeschreibung verzichtet werden kann. Nachdem die erste Version vorgestellt wurde, wird diese sicher auch verstanden. Es handelt sich um eine "gemäßigte" Fassung, die insbesondere keine spezielle Variablenübergabe an Subroutinen benötigt.

Die dritte Version kommt ganz ohne Unterprogramm aus, wir halten allerdings auch dies nicht für die beste Lösung.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM   **** DEMONSTRATIONSBEISPIEL ***
11  REM    ** FUER UNTERPROGRAMMTECHNIK **
200  REM  BEGINN DES PROGRAMMS
210 N = 20
220  TEXT : HOME : DIM N$(N),A$(N)
230 M = 5
240  FOR I = 1 TO N: READ N$(I),A$(I): NEXT
245  FOR Q = 1 TO 40: PRINT "*";: NEXT : PRINT : HTAB 6:
PRINT "S O R T I E R P R O G R A M M": PRINT : FOR Q =
1 TO 40: PRINT "*";: NEXT : PRINT : POKE 34,6: HOME
250  VTAB 12: HTAB 17: INVERSE : PRINT "1. RUNDE": NORMAL

253  FOR I = 1 TO N - 1: FOR J = I + 1 TO N: IF N$(I) >
N$(J) THEN H$ = N$(I):N$(I) = N$(J):N$(J) = H$:H$ = A$(I):A$(I)
= A$(J):A$(J) = H$
256  NEXT J,I
260  HOME :Z = 0: FOR I = 1 TO N: PRINT N$(I),A$(I):Z =
Z + 1
263  IF Z = M OR I = N THEN Z = 0: PRINT : FOR Q = 1 TO
40: PRINT "*";: NEXT : PRINT : VTAB 22: CALL  - 868: HTAB
6: INVERSE : PRINT "DRUECKEN SIE IRGENDEINE TASTE": NORMAL
: POKE  - 16368,0: WAIT  - 16384,128: POKE  - 16368,0:
HOME
266  NEXT I
270  VTAB 12: HTAB 17: INVERSE : PRINT "2. RUNDE": NORMAL

273  FOR I = 1 TO N - 1: FOR J = I + 1 TO N: IF A$(I) >
A$(J) THEN H$ = N$(I):N$(I) = N$(J):N$(J) = H$:H$ = A$(I):A$(I)
= A$(J):A$(J) = H$
276  NEXT J,I
```

```
280  HOME :Z = 0: FOR I = 1 TO N: PRINT N$(I),A$(I):Z =
Z + 1
283  IF Z = M OR I = N THEN Z = 0: PRINT : FOR Q = 1 TO
40: PRINT "*";: NEXT : PRINT : VTAB 22: CALL  - 868: HTAB
6: INVERSE : PRINT "DRUECKEN SIE IRGENDEINE TASTE": NORMAL
: POKE  - 16368,0: WAIT  - 16384,128: POKE  - 16368,0:
HOME
286  NEXT I
290  VTAB 12: HTAB 15: INVERSE : PRINT "E N D E": NORMAL
: POKE 34,0: PRINT : END
1000  DATA     MUELLER,40,FRIEDRICHS,23,MEIER,17,MAIER,52
1010  DATA   KOCH,45,SCHUHMANN,34,NEUHAUS,21,MIES,57
1020  DATA  LUDWIG,14,HAUSMANN,33,ADAM,41,GRIMM,28
1030  DATA  OBERMANN,62,GROTE,24,NAGEL,58,MUTH,46
1040  DATA  FROEHLICH,11,SCHMITT,43,KLEIN,65,MARSCH,19
```

Wieder wurde durch die Wahl der Zeilennummern versucht, Vergleiche zwischen den 3 Versionen zu erleichtern.

Prinzipiell sind sicher Programme, die mehr der Version 2 ähneln, den anderen Typen vorzuziehen.
Auch in unserem kleinen Beispiel stellt sich der Typ 2 als optimal hinsichtlich des Platzbedarfs dar (ohne REM-Statements im Hauptprogramm wäre der Unterschied noch deutlicher ausgefallen).
Natürlich sind die Differenzen in einem so einfachen Beispiel nicht besonders groß, aber es bestehen welche, wie folgende Tabelle zeigt:

Adresse für:	Speicher-stellen	Versionen		
		1	2	3
Programmanfang	1Ø3/1Ø4	2Ø49	2Ø49	2Ø49
Programmende	1Ø5/1Ø6	3123	31Ø5	3173
Speicherplatz in byte	-	1Ø75	1Ø57	1125
Ende benutzter Speicher	1Ø9/11Ø	3459	3294	3362
Speicherplatz für Variable	-	336	189	189
Gesamtplatz	-	1411	1246	1314

Man sieht, daß bei Version 1 trotz der vielen Unterprogramme mehr Platz benötigt wird als bei den beiden anderen; dies liegt vor allem an dem benötigten Platz für die Variablen. Aber auch das Programm allein ist etwas länger als die Version 2, da die Einsparungen durch Unterprogramme durch die aufwendige Variablenübergabe kompensiert werden. Bei längeren Unterprogrammen, die häufiger angesprungen werden, ist die Ersparnis natürlich größer. Gelegentlich stößt man bei sehr großen Programmen an die Kapazitätsgrenzen des APPLE, wenn man nicht eine so ausgeprägte Unterprogrammtechnik verwendet.
Doch sollten diese Beispiele vor allem auch deutlich machen, daß das Verständnis des Programmablaufs wesentlich vom Programmierstil abhängt.

5.3 Menütechnik

Viele Programme erfüllen nicht nur eine Aufgabe, sondern mehrere, und zwar nacheinander oder wahlweise. Für den zweiten Fall muß das Programm dem Benutzer zunächst einmal seine Möglichkeiten vorstellen, d.h. eine Übersicht über die gelösten Problembereiche. So kann etwa ein Textverarbeitungsprogramm neue Texte speichern, alte verändern oder erweitern. Diese Auswahlmöglichkeiten oder die entsprechende Bildschirmanzeige nennt man Menü.
Dabei handelt es sich also um die "Speisekarte" des Rechners. Auch der Catalog einer Diskette ist im Grunde nichts anderes, doch erwartet man von einem richtigen Menü eine ansprechende Gestaltung und eine einfache Methode der Auswahl. Je nach Wahl des Benutzers wird nun ein anderer Programmteil angesprungen. Dabei benötigt man also bedingte Sprunganweisungen. Außer der schon bekannten IF...THEN-Anweisung kennt APPLESOFT-BASIC noch zwei weitere bedingte Sprunganweisungen, eine davon mit "automatischer Rückkehr" durch RETURN.
Die Anweisungen lauten allgemein:

ON variable GOTO nr1, nr2, nr3, ...
ON variable GOSUB nr1, nr2, nr3, ...

Der Unterschied liegt nur in der Rückkehr nach ON... GOSUB aufgrund des Befehls RETURN (genau wie beim unbedingten GOSUB), die nach ON...GOTO nicht erfolgt. Der Befehl

```
ON  K  GOTO 1ØØ, 11Ø, 12Ø, 15Ø
```

hat unterschiedliche Auswirkungen bei unterschiedlichen Werten von K. Zunächst ist zu sagen, daß eventuell aufgetretene Nachkommastellen des K-Wertes einfach abgeschnitten werden. Hat K nach den Wert 1, so erfolgt ein Sprung zu Zeile 1ØØ, also in die erste hinter GOTO stehende Zeilennummer; für K=2 wird zu Zeile 11Ø gesprungen, für K=3 zu Zeile 12Ø, für K=4 zu Zeile 15Ø. Für alle Werte von K kleiner als 1 oder größer gleich 5 erfolgt kein Sprung; es wird der nächste Befehl bearbeitet, der aber auch in derselben Zeile folgen kann. Hier liegt also ein wesentlicher Unterschied zum IF...THEN-Befehl.

> Für Programmierer, die möglichst viele Befehle in eine Zeile schreiben wollen, da ja jede Zeilennummer Speicherplatz kostet, sei hier ein kleiner Trick eingeschoben:
> Wen es ärgert, daß etwa hinter dem Befehl IF A=1 THEN 5Ø keine weiteren Befehle mehr in der Zeile folgen können, kann stattdessen die Anweisung ON A GOTO 5Ø verwenden. Für A=1 erfolgt der gewünschte Sprung , falls A≠1 ist, werden die nachfolgenden Anweisungen derselben Zeile bearbeitet.

Für ON...GOSUB gilt genau das gleiche: nach der Bearbeitung des Unterprogramms folgt der Rücksprung zu dem folgenden Befehl - gleich ob er in derselben oder in der nachfolgenden Zeile steht.

In einem einfachen Anwendungsbeispiel soll verdeutlicht werden, was unter Menütechnik verstanden wird. Es handelt sich um die Bildschirmausgabe der Zahlen 1 bis 16. Das Programm zeigt eine 4*4 Matrix, die jeweils die Zahlen 1 bis 4, 5 bis 8, 9 bis 12 und 13 bis 16 in einer Zeile ausgibt. Dies kann auf 4 verschiedene Arten geschehen: in der üblichen Weise, d.h. 1 2 3 4 in der 1. Zeile, 5 6 7 8 in der zweiten usw.; in "gespiegelter" Form, d.h. 4 3 2 1 in der 1. Zeile, 8 7 6 5 in der zweiten usw.; in der normalen Weise mit Hervorhebung der Hauptdiagonalen und in der normalen Weise mit Hervorhebung der Nebendiagonalen. Das Menü bietet also vier Gänge:

```
*********************************
*                               *
*  DARSTELLUNG EINER 4*4-MATRIX *
*                               *
*********************************


          NORMALDARSTELLUNG..1

          SPIEGELMATRIX......2

          HAUPTDIAGONALE.....3

          NEBENDIAGONALE.....4

          EINGABE (5-ENDE) ..


     WAEHLEN SIE DURCH EINGABE DER ZIFFER !
```

Nach Eingabe einer Zahl zwischen 1 und 5 erfolgt die entsprechende Bildschirmanzeige aufgrund des folgenden Programms:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  DIM Z(4,4),Z$(4,4)
20  DATA  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
30  FOR I = 1 TO 4: FOR J = 1 TO 4: READ Z(I,J): NEXT :
NEXT
40  FOR I = 1 TO 4: FOR J = 1 TO 4:Z$(I,J) =  STR$ (Z(I,J)):
IF Z(I,J) < 10 THEN Z$(I,J) = " " + Z$(I,J)
50  NEXT J,I
60  HOME : HTAB 4: FOR Q = 1 TO 34: PRINT "*";: NEXT :
PRINT : HTAB 4: PRINT "*";: HTAB 37: PRINT "*"
70  HTAB 4: PRINT "*";: HTAB 7: PRINT "DARSTELLUNG EINER
4*4-MATRIX";: HTAB 37: PRINT "*"
80  HTAB 4: PRINT "*";: HTAB 37: PRINT "*": HTAB 4: FOR
I = 1 TO 34: PRINT "*";: NEXT : POKE 34,6
100  HOME : VTAB 10: HTAB 11: PRINT "NORMALDARSTELLUNG..1":
PRINT : HTAB 11: PRINT "SPIEGELMATRIX......2": PRINT :
HTAB 11: PRINT "HAUPTDIAGONALE.....3": PRINT : HTAB 11:
PRINT "NEBENDIAGONALE.....4": PRINT
110  VTAB 22: HTAB 3: INVERSE : PRINT "WAEHLEN SIE DURCH
EINGABE DER ZIFFER !";: NORMAL
120  VTAB 18: HTAB 11: INPUT "EINGABE (5-ENDE) ..";X$:X
=  VAL (X$)
130  FOR I = 1 TO 5: IF X <  > I THEN  NEXT : VTAB 18:
HTAB 30: INVERSE : PRINT "FEHLER";: NORMAL : FOR Q = 1
TO 1500: NEXT : HTAB 30: CALL  - 868: GOTO 120
140  ON X GOTO 500,520,540,570,230
200  VTAB 23: HTAB 6: INVERSE : PRINT "DRUECKEN SIE IRGENDEINE
TASTE": NORMAL : POKE  - 16368,0
210  GET A$: IF A$ = "" THEN 210
220  POKE  - 16368,0: GOTO 100
230  POKE 34,0: CALL  - 958: END
500  REM   AUSDRUCK DER MATRIX
510  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 1 TO 4: HTAB
(6 * J + 5): PRINT Z$(I,J);: NEXT : PRINT : PRINT : NEXT
: GOTO 200
520  REM   AUSDRUCK DER GESPIEGELTEN MATRIX
530  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 4 TO 1 STEP
 - 1: HTAB (35 - 6 * J): PRINT Z$(I,J);: NEXT : PRINT :
PRINT : NEXT : GOTO 200
540  REM   HERVORHEBUNG DER HAUPTDIAGONALE
550  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 1 TO 4: HTAB
(6 * J + 5): IF I = J THEN  INVERSE
560  PRINT Z$(I,J);: NORMAL : NEXT : PRINT : PRINT : NEXT
: GOTO 200
570  REM   HERVORHEBUNG DER NEBENDIAGONALE
580  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 1 TO 4: HTAB
(6 * J + 5): IF (I + J) = 5 THEN  INVERSE
590  PRINT Z$(I,J);: NORMAL : NEXT : PRINT : PRINT : NEXT
: GOTO 200
```

Das Programm soll kurz erläutert werden:

Sätze	Inhalt
1Ø	Es werden zwei doppelt indizierte Felder für die 4*4-Matrix dimensioniert.
2Ø-3Ø	DATA-Zeile und Einlesen der Daten
4Ø-5Ø	Die Zahlen werden in Strings verwandelt, die einstelligen dabei durch ein Leerzeichen ebenfalls auf zwei Stellen erweitert.
6Ø-8Ø	Ausgabe einer Überschrift mit anschließendem Sperren des entsprechenden Bildschirmbereichs.
1ØØ-12Ø	Menü - Die Auswahl muß über eine der Ziffern 1 bis 5 erfolgen.
13Ø	Für andere Werte erfolgt eine Fehlermeldung und die Eingabe wird erneut angefordert durch Wiederholung der gesamten Zeile 12Ø.
14Ø	Verzweigung durch ON...GOTO.
2ØØ-22Ø	Bedingte Warteschleifen, um dem Benutzer die Betrachtung des Bildschirmausdrucks zu ermöglichen.
23Ø	Bei der Eingabe 5 für "Ende des Programms" erfolgt die Aufhebung des Textfensters, die Bildschirmanzeige "WÄHLEN SIE DURCH EINGABE DER ZIFFER!" wird gelöscht und das Programm beendet

Sätze	Inhalt
5ØØ-51Ø	Die Zahlen 1 bis 16 werden in der üblichen Weise einer 4*4-Matrix ausgegeben mit Hilfe eines berechneten HTAB-Wertes. Für die Ausgabe werden die Strings Z$ gewählt, um die rechtsbündige Ausgabe zu sichern. Danach erfolgt der Sprung zur bedingten Warteschleife in Zeile 2ØØ.
52Ø-53Ø	Die Ausgabe erfolgt jeweils in den einzelnen Zeilen in umgekehrter Reihenfolge; deshalb hat nun die J-Schleife, die den Index für die Spalte liefert, negative Schrittweite. Folgt Sprung in Zeile 2ØØ.
54Ø-56Ø	In der üblichen Matrixform werden die Elemente der Hauptdiagonalen durch inverse Bildschirmausgabe gekennzeichnet.
57Ø-59Ø	Die Elemente der Nebendiagonale werden durch inverse Ausgabe hervorgehoben.

Die Auswahl aus dem Menü erfolgte in diesem Beispiel mit Hilfe der angegebenen Ziffern 1 bis 5. Die Eingabevariable ist X$, damit versehentlich eingegebene Buchstaben nicht die Rechnermeldung ?REENTER verursachen, die die Bildschirmaufteilung stören würde. Durch die VAL-Funktion werden solche Eingaben zu einem Wert von Null umgewandelt. Ein solcher Wert von X führt über die Zeile 130 zu einer Fehlermeldung, die in die übrige Bildschirmanzeige eingepaßt ist.
Die Verzweigung erfolgt über das ON...GOTO-Statement, stattdessen hätte auch der ON...GOSUB-Befehl gewählt werden können, da alle 4 Programmsegmente in den Zeilen 500 bis 590 durch die Sprunganweisung GOTO 200 beendet werden, d.h. nach den jeweiligen Anzeigen wird die Zeile hinter der Verzweigung bearbeitet. Dies wäre bei RETURN ebenfalls geschehen.

Bei der zweiten Form einer Menüauswahl wird die Auswahl über die Eingabe von Buchstaben erreicht. Zulässig sind die Buchstaben A bis E, die durch Zeile 40 in ein Stringarray B$ eingelesen werden aus der DATA-Zeile 600. Der zugehörige Arrayindex liefert gleichzeitig die numerische Variable, die für die Verzweigung durch ON...GOTO erforderlich ist.
Die Bildschirmanzeige des Menüs ist entsprechend geändert (Zeilen 100 bis 120).

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  DIM Z(4,4),Z$(4,4),B$(5)
20  DATA  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
30  FOR I = 1 TO 4: FOR J = 1 TO 4: READ Z(I,J): NEXT :
NEXT
40  FOR I = 1 TO 5: READ B$(I): NEXT
50  FOR I = 1 TO 4: FOR J = 1 TO 4:Z$(I,J) =  STR$ (Z(I,J)):
IF Z(I,J) < 10 THEN Z$(I,J) = " " + Z$(I,J)
60  NEXT J,I
70  HOME : HTAB 4: FOR Q = 1 TO 34: PRINT "*";: NEXT :
PRINT : HTAB 4: PRINT "*";: HTAB 37: PRINT "*"
80  HTAB 4: PRINT "*";: HTAB 7: PRINT "DARSTELLUNG EINER
4*4-MATRIX";: HTAB 37: PRINT "*"
90  HTAB 4: PRINT "*";: HTAB 37: PRINT "*": HTAB 4: FOR
I = 1 TO 34: PRINT "*";: NEXT : POKE 34,6
100  HOME : VTAB 10: HTAB 11: PRINT "NORMALDARSTELLUNG..A":
PRINT : HTAB 11: PRINT "SPIEGELMATRIX......B": PRINT :
HTAB 11: PRINT "HAUPTDIAGONALE.....C": PRINT
110  HTAB 11: PRINT "NEBENDIAGONALE.....D": PRINT : HTAB
11: PRINT "ENDE DES PROGRAMMS.E"
120  VTAB 20: HTAB 16: INVERSE : PRINT "WAEHLEN SIE": PRINT
: HTAB 6: NORMAL : PRINT "DURCH EINGABE DER BUCHSTABEN
!";
130  POKE  - 16368,0: GET X$: POKE  - 16368,0
140  FOR I = 1 TO 5: IF X$ <  > B$(I) THEN  NEXT I: VTAB
18: HTAB 32: INVERSE : PRINT "FEHLER";: NORMAL : FOR Q
= 1 TO 1500: NEXT Q: HTAB 32: CALL  - 868: GOTO 120
150  POKE 37,(6 + 2 * I): PRINT : POKE 36,29: FLASH : PRINT
B$(I): FOR Q = 1 TO 1500: NEXT : NORMAL
160  ON I GOTO 500,520,540,570,230
200  VTAB 23: HTAB 6: INVERSE : PRINT "DRUECKEN SIE IRGENDEINE
TASTE": NORMAL : POKE  - 16368,0
210  GET A$: IF A$ = "" THEN 210
220  POKE  - 16368,0: GOTO 100
230  POKE 34,0: POKE 37,16: PRINT : POKE 36,29: PRINT "E":
CALL  - 958: END
500  REM  AUSDRUCK DER MATRIX
510  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 1 TO 4: HTAB
(6 * J + 5): PRINT Z$(I,J);: NEXT : PRINT : PRINT : NEXT
: GOTO 200
520  REM  AUSDRUCK DER GESPIEGELTEN MATRIX
530  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 4 TO 1 STEP
 - 1: HTAB (35 - 6 * J): PRINT Z$(I,J);: NEXT : PRINT :
PRINT : NEXT : GOTO 200
540  REM  HERVORHEBUNG DER HAUPTDIAGONALE
550  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 1 TO 4: HTAB
(6 * J + 5): IF I = J THEN  INVERSE
560  PRINT Z$(I,J);: NORMAL : NEXT : PRINT : PRINT : NEXT
: GOTO 200
570  REM  HERVORHEBUNG DER NEBENDIAGONALE
580  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 1 TO 4: HTAB
(6 * J + 5): IF (I + J) = 5 THEN  INVERSE
590  PRINT Z$(I,J);: NORMAL : NEXT : PRINT : PRINT : NEXT
: GOTO 200
600  DATA  A,B,C,D,E
```

Die Eingabe erfolgt nun über GET X$ (Zeile 130), unzulässige Eingaben werden über den Vergleich von X$ und B$(I) zurückgewiesen bei entsprechender Meldung. Diese Zeile 140 ist etwas verschieden von der Zeile 130 des vorigen Programms, Zeile 150 ist etwas völlig Neues:
Wenn in der Schleife in der Zeile 140 der Index des richtig ausgewählten Buchstabens bestimmt worden ist, wird der Cursor auf den entsprechenden Buchstaben des Menüs positioniert durch die zwei POKE-Befehle für die Speicherstellen 37 und 36 und der Buchstabe beginnt zu blinken. Nach einer leeren Schleife zum Warten erfolgt dann die Verzweigung durch Zeile 160. Das übrige Programm ist identisch mit der ersten Version.

Eine dritte Version des Programms unterscheidet sich wiederum nur durch die Art der Auswahl von den zwei vorhergehenden, also nur in den Zeilen 100 bis 150, insbesondere durch die Zeilen 130 bis 140, die die Bestimmung des I-Wertes für die Verzweigung enthalten. Durch die P-Schleife in Zeile 130 wandert der Cursor links neben dem Menü von Zeile zu Zeile, immer wieder bei der ersten Auswahlmöglichkeit beginnend.
Dies geschieht solange, bis irgendeine Taste gedrückt wird. Die Prüfung der Speicherstelle -16384 mußte deshalb in die Schleife aufgenommen werden.
Aus der Position P, an der der Cursor sich zu diesem Zeitpunkt befindet, wird dann in Zeile 140 der entsprechende I-Wert berechnet.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  DIM Z(4,4),Z$(4,4)
20  DATA   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
30  FOR I = 1 TO 4: FOR J = 1 TO 4: READ Z(I,J): NEXT :
NEXT
40  FOR I = 1 TO 4: FOR J = 1 TO 4:Z$(I,J) =  STR$ (Z(I,J)):
IF Z(I,J) < 10 THEN Z$(I,J) = " " + Z$(I,J)
50  NEXT J,I
60  HOME : HTAB 4: FOR Q = 1 TO 34: PRINT "*";: NEXT :
PRINT : HTAB 4: PRINT "*";: HTAB 37: PRINT "*"
70  HTAB 4: PRINT "*";: HTAB 7: PRINT "DARSTELLUNG EINER
4*4-MATRIX";: HTAB 37: PRINT "*"
80  HTAB 4: PRINT "*";: HTAB 37: PRINT "*": HTAB 4: FOR
I = 1 TO 34: PRINT "*";: NEXT : POKE 34,6
100  HOME : VTAB 10: HTAB 11: PRINT "1. NORMALDARSTELLUNG":
PRINT : HTAB 11: PRINT "2. SPIEGELMATRIX": PRINT : HTAB
11: PRINT "3. HAUPTDIAGONALE": PRINT
110  HTAB 11: PRINT "4. NEBENDIAGONALE": PRINT : HTAB 11:
PRINT "5. ENDE DES PROGRAMMS"
120  VTAB 20: HTAB 16: INVERSE : PRINT "WAEHLEN SIE": PRINT
: HTAB 4: NORMAL : PRINT "DURCH DRUECKEN IRGENDEINER TASTE
!";: POKE  - 16368,0
130  FOR P = 10 TO 18 STEP 2: VTAB P: HTAB 2: INVERSE :
PRINT " ";: FOR Q = 1 TO 750: NEXT Q: IF  PEEK ( - 16384)
< 128 THEN  HTAB 2: NORMAL : PRINT " ": NEXT P: GOTO 130
140 I = P / 2 - 4: NORMAL : HTAB 2: PRINT " "
150  ON I GOTO 500,520,540,570,230
200  VTAB 23: HTAB 6: INVERSE : PRINT "DRUECKEN SIE IRGENDEINE
TASTE": NORMAL : POKE  - 16368,0
210  GET A$: IF A$ = "" THEN 210
220  POKE  - 16368,0: GOTO 100
230  POKE  - 16368,0: POKE 34,0: CALL  - 958: END
500  REM   AUSDRUCK DER MATRIX
510  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 1 TO 4: HTAB
(6 * J + 5): PRINT Z$(I,J);: NEXT : PRINT : PRINT : NEXT
: GOTO 200
520  REM   AUSDRUCK DER GESPIEGELTEN MATRIX
530  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 4 TO 1 STEP
 - 1: HTAB (35 - 6 * J): PRINT Z$(I,J);: NEXT : PRINT :
PRINT : NEXT : GOTO 200
540  REM   HERVORHEBUNG DER HAUPTDIAGONALE
550  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 1 TO 4: HTAB
(6 * J + 5): IF I = J THEN  INVERSE
560  PRINT Z$(I,J);: NORMAL : NEXT : PRINT : PRINT : NEXT
: GOTO 200
570  REM   HERVORHEBUNG DER NEBENDIAGONALE
580  HOME : VTAB 10: FOR I = 1 TO 4: FOR J = 1 TO 4: HTAB
(6 * J + 5): IF (I + J) = 5 THEN  INVERSE
590  PRINT Z$(I,J);: NORMAL : NEXT : PRINT : PRINT : NEXT
: GOTO 200
```

Eine Menüauswahl kann immer dann erfolgen, wenn ein Programm mehrere Aufgaben erfüllt - wahlweise oder nacheinander in beliebiger Reihenfolge. Für die Gestaltung gibt es verschiedene Möglichkeiten, wie die drei Demonstrationsprogramme gezeigt haben. Sie können dem Leser vielleicht als Anregung dienen für die Entwicklung weiterer Formen einer Menüauswahl.

5.4 Mischen von Programmteilen

Häufig benutzte Unterprogramme möchte man nicht immer wieder neu schreiben, sondern ohne großen Aufwand bei den aktuellen Programmieraufgaben verwenden. Eine Möglichkeit dazu bieten die sogenannten EXEC-Dateien. Das sind sequentielle Textfiles, die Kommandos und Programmzeilen enthalten können. Ihre Erstellung ist etwas umständlich, deshalb sollen einige einfache Beispiele zunächst das Vorgehen verdeutlichen.

Wenn man sich über den Platzbedarf eines Programms informieren will, müssen eine ganze Reihe von Speicherstellen abgefragt werden. Liegt der Sinn dieser Abfragen im Vergleich des erforderlichen Speicherplatzes mehrerer Programmversionen, wie etwa im Abschnitt 5.2 demonstriert, so ist es etwas lästig, immer wieder dieselbe Befehlsfolge eingeben zu müssen. Man kann diese Befehle auch in einer Textdatei abspeichern und dann durch das Kommando EXEC immer wieder abrufen.

Das folgende Programm mit dem Namen MAKE SPEICHER leistet diese Abspeicherung.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10 D$ =  CHR$ (4): REM   CTRL-D FUER DISKETTENADRESSIERUNG
15 Q$ =  CHR$ (34): REM  ANFUEHRUNGSSTRICHE
20  PRINT D$;"OPEN SPEICHER"
30  PRINT D$;"WRITE SPEICHER"
40  PRINT "?";Q$;"103: ";Q$;";PEEK(103)+PEEK(104)*256"
50  PRINT "?";Q$;"105: ";Q$;";PEEK(105)+PEEK(106)*256"
60  PRINT "?";Q$;"109: ";Q$;";PEEK(109)+PEEK(110)*256"
70  PRINT "?";Q$;"111: ";Q$;";PEEK(111)+PEEK(112)*256"
80  PRINT "?";Q$;"115: ";Q$;";PEEK(115)+PEEK(116)*256"
90  PRINT D$;"CLOSE"
100  END
```

Wird es durch RUN gestartet, so wird ein sequentielles Textfile mit dem Namen SPEICHER geöffnet zum Schreiben. Die PRINT-Befehle der Zeilen 4Ø bis 8Ø bewirken dann keine Bildschirmanzeige, sondern die Abspeicherung auf der Diskette. Abgespeichert werden die in Anführungszeichen eingeschlossenen Befehle. Es handelt sich wiederum um PRINT-Statements, und zwar liefert jede Befehlszeile eine Bildschirmanzeige über eine Speicheradresse und ihren Inhalt, dabei müssen für die Erstellung der Befehlszeile die sonst üblichen Anführungszeichen durch die entsprechende CHR$-Funktion ersetzt werden, um Mißverständnisse zu vermeiden.

Nach Durchführung des Programms MAKE SPEICHER stehen die folgenden Zeilen im Textfile SPEICHER auf der Diskette:

```
PRINT "103: ";PEEK(103)+PEEK(104)*256
PRINT "105: ";PEEK(105)+PEEK(106)*256
PRINT "109: ";PEEK(109)+PEEK(110)*256
PRINT "111: ";PEEK(111)+PEEK(112)*256
PRINT "115: ";PEEK(115)+PEEK(116)*256
```

Nachdem nun irgendein Programm geladen worden ist, kann durch EXEC SPEICHER die Untersuchung des Platzbedarfs erfolgen. Die Wirkung der Kommandos ist dieselbe, als wenn sie gerade über die Tastatur eingegeben worden wären.

Das zweite Beispiel erstellt eine Textdatei DRUCK, die sowohl Kommandos als auch Programmzeilen enthält.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10 D$ =  CHR$ (4):Q$ =  CHR$ (34)
20  PRINT D$;"OPEN DRUCK"
30  PRINT D$;"WRITE DRUCK"
40  PRINT "3 HOME:HTAB(9):PRINT ";Q$;"APPLE-TIPS UND TRICKS";Q$
50  PRINT "4 PRINT:PRINT:HTAB(9):PRINT ";Q$;"R.PRUST /
W.VOSS 1984";Q$;":PRINT"
60  PRINT "PR#1"
70  PRINT "PRINT  CHR$ (9);";Q$;"80N";Q$
80  PRINT " PRINT  CHR$ (9);";Q$;"10L";Q$
90  PRINT "PRINT CHR$(9);";Q$;"65R";Q$
100  PRINT "PRINT CHR$(9);";Q$;"65P";Q$
110  PRINT "LIST 3-"
120  PRINT "PR#0"
130  PRINT D$;"CLOSE DRUCK"
140  END
```

Durch die Zeilen 4Ø und 5Ø werden zwei Programmzeilen mit den Nummern 3 und 4 in die Datei DRUCK geschrieben. Wieder müssen die Anführungsstriche der PRINT-Befehle durch CHRØ(34) ersetzt werden. Die Zeilen 6Ø bis 12Ø erzeugen Kommandozeilen im Textfile, dessen Inhalt also aus zwei Programmzeilen und einigen Druckerkommandos besteht:

```
3 HOME:HTAB(9):PRINT "APPLE-TIPS UND TRI
CKS"
4 PRINT:PRINT:HTAB(9):PRINT "R.PRUST / W
.VOSS 1984":PRINT
PR#1
PRINT  CHR$ (9);"80N"
 PRINT  CHR$ (9);"10L"
PRINT CHR$(9);"65R"
PRINT CHR$(9);"65P"
LIST 3-
PR#0
```

Wird diese Datei durch EXEC DRUCK aufgerufen, werden zunächst die Zeilen 3 und 4 dem vorher geladenen Programm hinzugefügt; dabei wird nicht - wie bei RUN - das übrige Programm gelöscht. Anschließend werden die Druckereinstellungen vorgenommen und das erweiterte Programm wird auf dem Drukker gelistet.

Eine andere, etwas komfortablere Möglichkeit, komplette Programmlistings auf dem Drucker erstellen zu lassen und Zeilen zu einem vorhandenen Programm hinzuzufügen, bietet MAKE LIST:

```
1  GOTO 10
2  PR# 1: PRINT  CHR$ (9);"80N": PRINT  CHR$ (9);"10L":
PRINT  CHR$ (9);"65R": PRINT  CHR$ (9);"65P": LIST 3 -
: PR# 0: END
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10 D$ =  CHR$ (4)
20  PRINT D$;"OPEN LIST"
30  PRINT D$;"WRITE LIST"
40  LIST 2 - 4
50  PRINT "RUN"
60  PRINT "DEL 2,4"
70  PRINT D$;"CLOSE LIST"
80  TEXT : END
```

Es wird ein Textfile LIST erzeugt mit folgendem Inhalt:

```
2  PR# 1: PRINT  CHR$ (9);"80N": PRINT
     CHR$ (9);"10L": PRINT  CHR$
    (9);"65R": PRINT  CHR$ (9);"
    65P": LIST 3 - : PR# 0: END

3  HOME : HTAB (9): PRINT "APPLE-
     TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT
     "R.PRUST / W.VOSS 1984": PRINT

RUN
DEL 2,4
```

Nach dem Kommando EXEC List werden dem im Arbeitsspeicher vorhandenen Programm die Zeile 2 bis 4 hinzugefügt; Voraussetzung ist, daß das ursprüngliche Programm nur Zeilennummern enthält, die größer sind als 4. Dann wird durch RUN das erweiterte Programm gestartet, d.h. es wird Zeile 2 mit den Druckeranweisungen ausgeführt. Danach geht die Ablaufkontrolle wieder an LIST zurück, das als letzten Befehl das Löschen der Programmzeilen 2 bis 4 durchführt.

Falls also während der Programmierarbeit ein Druckerlisting erforderlich ist, z.B. zur Fehlersuche, gibt man EXEC LIST ein; es wird ein Listing mit zwei zusätzlichen Zeilen durchgeführt und anschließend steht das Programm in der vorherigen Form für die weitere Bearbeitung im Arbeitsspeicher zur Verfügung.

Eine weitere Möglichkeit, Textfiles nützlich einzusetzen, besteht im Zusammenmischen von mehreren Programmteilen, die schon bei früheren Programmieraufgaben benötigt wurden. Es gibt einige Unterprogramme, deren Aufgabenstellung immer wieder auftaucht - genau wie das Drucken eine immer wieder erforderliche Arbeit ist. Für solche häufig auftretenden Probleme lohnt es sich, ein Programm zu erstellen, das ein Textfile erzeugt mit den nötigen Programmzeilen, wobei die erste Zeilennummer des Unterprogramms jeweils beliebig gewählt werden kann. Das Vorgehen ist etwas umständlich, deshalb ist es bei wenig benötigten Subroutinen oft schneller, die entsprechenden Programmzeilen neu einzutippen.

Immer wieder taucht das Problem des Sortierens eines Arrays bei gleichzeitgem Mitführen eines zweiten Arrays auf, und zwar für numerische Arrays und Stringarrays. Deshalb sei es als Demonstrationsbeispiel gewählt. Folgendes Hauptprogramm befinde sich im Arbeitsspeicher:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  EINBAU EINES UNTERPROGRAMMS
20 N = 20:M = 5: TEXT : HOME : DIM N$(N),A$(N)
30  GOTO 100
40  VTAB 22: CALL  - 868: HTAB 6: INVERSE : PRINT "DRUECKEN
SIE IRGENDEINE TASTE": NORMAL : POKE  - 16368,0: WAIT
- 16384,128: POKE  - 16368,0: HOME : RETURN
50  FOR Q = 1 TO 40: PRINT "X";: NEXT : PRINT : RETURN

100  REM  BEGINN DES PROGRAMMS
110  FOR I = 1 TO N: READ N$(I),A$(I): NEXT
120  GOSUB 200
130  HOME :Z = 0: FOR I = 1 TO N: PRINT N$(I),A$(I):Z =
Z + 1: IF Z = M OR I = N THEN Z = 0: PRINT : GOSUB 50:
GOSUB 40: REM       STRICH UND WARTEN
140  NEXT I
150  VTAB 12: HTAB 15: INVERSE : PRINT "E N D E": NORMAL
: POKE 34,0: PRINT : END
```

Es handelt sich um ein Hauptprogramm mit zwei kleinen Unterprogrammen (Zeilen 4Ø bis 5Ø), das Werte in zwei Stringarrays einliest; Die Strings des einen Arrays sollen nach dem Alphabet sortiert werden. Alle Umstellungen innerhalb des einen Arrays sollen bei dem zweiten entsprechend erfolgen. Zu diesem Zweck soll in Zeile 12Ø das Unterprogramm ab Zeile 2ØØ dienen.
Es ist auf Diskette vorhanden in der Form:

```
10  FOR I = 1 TO N - 1: FOR J = I + 1 TO N: IF N$(I) >
N$(J) THEN H$ = N$(I):N$(I) = N$(J):N$(J) = H$:H$ = A$(I):A$(I)
= A$(J):A$(J) = H$
20  NEXT J,I
30  RETURN
```

Man hat nun erstens die Möglichkeit, die Zeilen mit anderen Zeilennummern neu einzutippen. Zweitens kann man mit Hilfe der ESC-Taste umnumerieren (siehe Kap. 6). Oder man entschließt sich, einmal etwas mehr Arbeit zu investieren und dadurch diese Subroutine für die Zukunft mit beliebigen Zeilennummern verfügbar zu haben. Zunächst wird das Hauptprogramm auf Diskette gespeichert. Dann tippt man das folgende Programm ein mit Namen MAKE SORTIEREN:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  ERSTELLUNG EINES UNTERPROGRAMMS
20  REM  MIT BELIEBIGER ZEILENNUMMER
30  INPUT "ERSTE ZEILENNUMMER ";ZN
40 D$ =  CHR$ (4)
50  PRINT D$;"OPEN SORTIEREN"
60  PRINT D$;"DELETE SORTIEREN"
70  PRINT D$;"OPEN SORTIEREN"
80  PRINT D$;"WRITE SORTIEREN"
90  PRINT ZN;" FOR I = 1 TO N - 1: FOR J = I + 1 TO N:
IF N$(I) > N$(J) THEN H$ = N$(I):N$(I) = N$(J):N$(J) =
H$:H$ = A$(I):A$(I) = A$(J):A$(J) = H$":ZN = ZN + 1
100  PRINT ZN;" NEXT J,I":ZN = ZN + 1
110  PRINT ZN;" RETURN"
120  PRINT D$;"CLOSE SORTIEREN"
130  END
```

Es fordert zunächst eine Eingabe für die erste Zeilennummer, die das Unterprogramm erhalten soll. Danach wird eine eventuell vorhandene alte Textdatei mit dem Namen SORTIEREN gelöscht und danach eine neue zum Schreiben eröffnet. Durch die PRINT-Befehle der Zeilen 9Ø bis 11Ø werden Programmzeilen in die Datei SORTIEREN geschrieben. Dabei wird die Variable ZN durch die aktuelle Zeilennummer ersetzt und durch ZN=ZN+1 für jede neue Programmzeile erhöht, so daß SORTIEREN folgenden Inhalt erhält:

```
200 FOR I = 1 TO N - 1: FOR J = I + 1 TO N: IF N$(I) >
N$(J) THEN H$ = N$(I):N$(I) = N$(J):N$(J) = H$:H$ = A$(I):A$(I)
= A$(J):A$(J) = H$
201 NEXT J,I
202 RETURN
```

Durch RUN bzw. RUN MAKE SORTIEREN (falls sich das Programm schon auf der Diskette befindet) wird die Erstellung des Textfiles SORTIEREN gestartet.

Danach wird das Hauptprogramm wieder in den Arbeitsspeicher geladen und durch das Kommando EXEC SORTIEREN um die Unterprogrammzeilen 2ØØ bis 2Ø2 ergänzt.

Das Schreiben der Programme, die die Textdateien erzeugen, ist etwas mühsam, insbesondere wenn PRINT-Befehle in die Zeilen des Textfiles aufgenommen werden sollen, da Anführungsstriche , die später im Textfile stehen sollen, im erzeugenden Programm durch CHR$(34) ersetzt werden müssen. Dies war schon den ersten beiden Beispielen dieses Abschnitts zu entnehmen. Ein sehr umfangreiches Beispiel für den Einsatz von EXEC-Dateien findet sich im Abschnitt 7.5.

Man kann dies Verfahren auch einsetzen, um sich beispielsweise die Gestaltung eines Menus zu vereinfachen. Die erforderlichen Programmzeilen sind ja immer ähnlich aufgebaut, so daß sich die Verwendung eines "Programmzeilengenerators" unmittelbar anbietet.

Der wesentliche Nachteil liegt im mehrfachen Abspeichern und Laden; man kann nicht durchgängig am Hauptprogramm arbeiten; denn ohne vorheriges Sichern wird durch RUN MAKE SORTIEREN oder ein entsprechendes Kommando für andere Programmzeilengeneratoren das Hauptprogramm zerstört.
Allerdings kann man einen Teil dieser Aufgaben einer Textdatei übertragen. Dies wird an einem umfangreichen Beispiel, nämlich der Erstellung eines Druckerlisting, vorgeführt, das zugleich die Möglichkeiten und die Umständlichkeit der Gestaltung von EXEC-Dateien deutlich macht.

Ausgangssituation sei ein Programm im Arbeitsspeicher, das ganz oder teilweise gelistet werden soll, ohne daß der Benutzer Kommandos über die Tastatur abschicken will. EXEC LISTER ist das letzte Kommando, das einzutippen ist. Es ruft ein Textfile auf, das von dem folgenden Programm MAKE LISTER erzeugt wurde:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10 D$ =  CHR$ (4):Q$ =  CHR$ (34)
20  PRINT D$;"OPEN LISTER"
30  PRINT D$;"DELETE LISTER"
40  PRINT D$;"OPEN LISTER"
50  PRINT D$;"WRITE LISTER
60  PRINT "Y=PEEK(175)+PEEK(176)*256"
70  PRINT "Q =  PEEK (103) +  PEEK (104) * 256:X1 =  PEEK
(Q + 2) +  PEEK (Q + 3) * 256"
80  PRINT "FOR Q=Q TO Y: Z =  PEEK (Q) +  PEEK (Q + 1)
* 256: IF  PEEK (Z) +  PEEK (Z + 1) * 256 <  > 0 THEN Q
= Z-1: NEXT Q"
90  PRINT "X2 =  PEEK (Q+2) +  PEEK (Q + 3) * 256"
100  PRINT "Z1 = X1:Z2 = X2"
110  PRINT "X =  PEEK (115) +  PEEK (116) * 256"
120  PRINT "T=INT(Z1/256):POKE X-4,T:POKE X-5,Z1-T*256"
130  PRINT "T=INT(Z2/256):POKE X-2,T:POKE X-3,Z2-T*256"
140  PRINT "T=INT((X-6)/256):POKE 116,T:POKE 115,X-6-T*256"
150  PRINT "EXEC LST"
160  PRINT  CHR$ (4);"CLOSE LISTER"
170  END
```

Die Textdatei enthält nachstehende Befehlsfolge:

```
Y=PEEK(175)+PEEK(176)*256
Q =  PEEK (103) +  PEEK (104) * 256:X1 =  PEEK (Q + 2)
+  PEEK (Q + 3) * 256
FOR Q=Q TO Y: Z =  PEEK (Q) +  PEEK (Q + 1) * 256: IF
PEEK (Z) +  PEEK (Z + 1) * 256 <  > 0 THEN Q = Z-1: NEXT
Q
X2 =  PEEK (Q+2) +  PEEK (Q + 3) * 256
Z1 = X1:Z2 = X2
X =  PEEK (115) +  PEEK (116) * 256
T=INT(Z1/256):POKE X-4,T:POKE X-5,Z1-T*256
T=INT(Z2/256):POKE X-2,T:POKE X-3,Z2-T*256
T=INT((X-6)/256):POKE 116,T:POKE 115,X-6-T*256
EXEC LST
```

Mit Hilfe von PEEK- und POKE-Befehlen werden zunächst die größte und kleinste Zeilennummer des Programms ermittelt (Z2 und Z1). Dabei wird die kleinste Zeilennummer Z1 an der 2. und 3. Speicherstelle hinter dem Programmanfang gefunden, der über die Adressen 1Ø3/1Ø4 ermittelt wird. Jede Zeile ist in folgender Weise abgespeichert: die ersten 2 bytes enthalten die Adresse für die nächste Programmzeile (das niederwertige byte zuerst), die nächsten 2 bytes enthalten die Zeilennummer, danach wird der Text abgespeichert. Dies wird nun ausgenutzt, um die größte Zeilennummer Z2 zu bestimmen. Mit Hilfe des Pointers zur nächsten Zeile

"springt" der Rechner schnell von Zeilennummer zu Zeilennummer, bis der Pointer auf eine Speicherstelle zeigt, die keine Adresse mehr enthält, sondern zusammen mit der nachfolgenden einen Wert von Null enthält. An der so ermittelten Stelle Q beginnt somit die Abspeicherung für die letzte Zeile, d.h. an den Stellen Q+2 und Q+3 findet sich die größte Zeilennummer des Programms, Z2 kann also ermittelt werden.

Damit diese Werte von Z1 und Z2 im weiteren Ablauf nicht verlorengehen, werden sie unmittelbar unterhalb von HIMEM (Speicherstellen 115/116) gespeichert, und zwar niederwertiges und höherwertiges byte von Z1, dann niederwertiges und höherwertiges byte von Z2 mit wachsenden Speicheradressen. Anschließend wird HIMEM herabgesetzt, um die Werte zu schützen.

Die letzte Anweisung des Textfiles LISTER ruft die EXEC-Datei LST auf. Auch diese sollte schon vorher durch das folgende Programm MAKE LST erstellt worden sein:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  ERSTELLUNG EINES EXEC'S
20  REM  FUER DRUCKERLISTINGS
30 D$ =  CHR$ (4)
40  PRINT D$;"OPEN LST"
50  PRINT D$;"WRITE LST"
60  PRINT "SAVE ZWISCHEN"
70  PRINT "RUN EINGABE"
80  PRINT "LOAD ZWISCHEN"
90  PRINT "DELETE ZWISCHEN"
100  PRINT "EXEC LSTR"
110  PRINT D$;"CLOSE LST"
```

Das hierdurch erstellte Textfile übernimmt die Steuerung des gesamten übrigen Ablaufs. Man sollte beachten, daß ein EXEC-Befehl in einem Textfile der letzte Befehl sein muß, da das erste Textfile geschlossen wird, bevor die Durchführung des zweiten beginnt.

LST enthält folgende Befehlsfolge:

```
SAVE ZWISCHEN
RUN EINGABE
LOAD ZWISCHEN
DELETE ZWISCHEN
EXEC LSTR
```

Zunächst wird das zu listende Programm auf der Diskette gesichert unter dem Namen ZWISCHEN , danach wird ein Programm EINGABE gestartet, das u.a. auch ein Textfile LSTR aus Programmzeilen erzeugt. Nun wird das ursprüngliche Programm wieder geladen und die temporäre Kopie wird gelöscht, danach wird die Textdatei LSTR aufgerufen, die folgende Befehle enthält:

```
122 PR#1:PRINT CHR$(9);"80N"
123 PRINT CHR$(9);"10L"
124 PRINT CHR$(9);"65R"
125 PRINT CHR$(9);"65P"
126 LIST 20-40: PR#0: END
RUN 122
DEL 122,126
```

Die im Textfile auftauchenden Zeilennummern sind vom zu listenden Programm abhängig und von möglichen Benutzereingaben. Durch den Befehl EXEC LSTR werden hinter das ursprüngliche Programmende 5 neue Zeilen angefügt mit den Druckeranweisungen. Dann wird das erweiterte Programm ab der ersten neuen Zeile gestartet, d.h. es erfolgt das Druckerlisting, anschließend werden die angefügten Zeilen wieder gelöscht, so daß unser Programm in seiner ursprünglichen Form im Arbeitsspeicher zur Verfügung steht. Die Zeilennummern 2Ø bis 4Ø, die gelistet werden sollen, sind nur ein Beispiel; die an dieser Stelle einzusetzenden Zeilennummern werden mit Hilfe des Programms EINGABE bestimmt, das auch das Textfile LSTR erstellt:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  ERSTELLUNG EINES DRUCK-EXEC'S
20 D$ =  CHR$ (4):Q$ =  CHR$ (34)
30 T =  PEEK (115) +  PEEK (116) * 256
40 Z1 =  PEEK (T + 1) +  PEEK (T + 2) * 256
50 Z2 =  PEEK (T + 3) +  PEEK (T + 4) * 256:X2 = Z2
60 TT =  INT ((T + 6) / 256): POKE 115,T + 6 - TT * 256:
POKE 116,TT
70  HOME : VTAB 5: PRINT "BEGINN DES AUSDRUCKS ";
80  FOR I = 1 TO 6: FLASH : PRINT  CHR$ (160);: POKE  -
16368,0: WAIT  - 16384,128
90  NORMAL : POKE 36, PEEK (36) - 1
100 X =  PEEK ( - 16384): IF X <  > 141 THEN X$ = X$ +
 CHR$ (X - 128): PRINT  CHR$ (X);: NEXT I
110  PRINT  CHR$ (160): POKE  - 16368,0: IF X$ <  > ""
THEN Z1 =  VAL (X$)
120  VTAB 8: PRINT "ENDE DES AUSDRUCKS   ";
130 X$ = "": FOR I = 1 TO 6: FLASH : PRINT  CHR$ (160);:
POKE  - 16368,0: WAIT  - 16384,128
140  NORMAL : POKE 36, PEEK (36) - 1
150 X =  PEEK ( - 16384): IF X <  > 141 THEN X$ = X$ +
 CHR$ (X - 128): PRINT  CHR$ (X);: NEXT
160  PRINT  CHR$ (160): POKE  - 16368,0: IF X$ <  > ""
THEN Z2 =  VAL (X$)
```

```
170  PRINT D$;"OPEN LSTR"
180  PRINT D$;"DELETE LSTR"
190  PRINT D$;"OPEN LSTR"
200  PRINT D$;"WRITE LSTR"
210  PRINT X2 + 2;" PR#1:PRINT CHR$(9);";Q$;"80N";Q$
220  PRINT X2 + 3;" PRINT CHR$(9);";Q$;"10L";Q$
230  PRINT X2 + 4;" PRINT CHR$(9);";Q$;"65R";Q$
240  PRINT X2 + 5;" PRINT CHR$(9);";Q$;"65P";Q$
250  PRINT X2 + 6;" LIST ";Z1;"-";Z2;": PR#0: END"
260  PRINT "RUN ";X2 + 2
270  PRINT "DEL ";X2 + 2;",";X2 + 6
280  PRINT D$;"CLOSE LSTR"
290  END
```

Hierzu ist vielleicht eine etwas ausführlichere Programmbeschreibung angebracht:

Sätze	Inhalt
2Ø	Festlegung der Stringkonstanten für die Textfile - Erstellung ab Zeile 17Ø.
3Ø	Die Variable T wird auf den Wert von HIMEM gesetzt, der ja durch LISTER reduziert worden ist.
4Ø-5Ø	In den 4 nachfolgenden Speicherstellen befinden sich die Werte von Z1 und Z2 für die kleinste und größte Zeilennummer.
6Ø	HIMEM wird auf den ursprünglichen Wert (T+6) zurückgesetzt. Die Variable X2 liefert später die Werte für die Zeilennummern der Datei LSTR.
7Ø-16Ø	Der Benutzer kann den Programmbereich angeben, der gelistet werden soll. Dies geht nicht über INPUT-Befehle, da bei Programmstart aus einem Textfile alle INPUT-Statements das Einlesen des nächsten Textfile-Feldes bewirken. Also muß direkt die Tastatur abgefragt werden. 7Ø Bildschirmanzeige 8Ø Öffnen einer Schleife für die Eingabe der Ziffern der gewünschten Zeilennummer. Einschließlich RETURN sind 6 Eingaben möglich. Um das Erscheinungsbild eines INPUT-Befehls zu simulieren, wird ein blinkendes Leerzeichen hinter die bisherige Bildschirmanzeige gesetzt. Danach hält das Programm an, bis eine Tastatureingabe erfolgt ist.

Sätze	Inhalt
	9Ø Für die Bildschirmanzeige der eingegebenen Zeichen wird auf NORMAL umgeschaltet und der Cursor in derselben Zeile um eine Stelle nach links versetzt (d.h. an die Stelle des blinkenden Zeichens).
	1ØØ Nun erfolgt die Überprüfung der Tastatureingabe auf RETURN. Solange andere Zeichen eingegeben werden (deren Code ungleich 141 ist), werden sie an einen String X$ angehängt. Dabei müssen wieder die unterschiedlichen ASC-Codezahlen berücksichtigt werden (siehe Kap. 3).
	11Ø Sobald ein Code von 141 erreicht wird, also RETURN gedrückt wurde, ist X$ vollständig. Das letzte blinkende Leerzeichen wird durch ein nomales ersetzt und der Tastaturpuffer freigemacht. Falls X$ dann nicht der leere String ist, wird die Variable Z1 für die erste Zeilennummer des Listings mit dem Inhalt von X$ gefüllt. Ansonsten bleibt es auf dem Wert der kleinsten Zeilennummer.
	12Ø-16Ø Eine entsprechende Befehlsfolge für die Eingabe der letzten zu listenden Zeile. Voreinstellung ist die letzte Programmzeile.
17Ø-28Ø	Erstellung des Textfiles LSTR Die Befehlsfolgen sind aus früheren Beispielen schon bekannt. Wichtig ist, daß die Variablen X2, Z1 und Z2 bei jedem Durchlauf des Programms EINGABE andere Werte haben können.

Mit diesem umfangreichen Beispiel soll dieser Abschnitt beendet werden. Da EXEC-Dateien meist sehr lange Ausführungszeiten benötigen, sollte der Leser sich das Beispiel nicht zum Vorbild nehmen; gleichwohl kann man ihm entnehmen, daß nahezu alle Befehle in Textfiles aufgenommen werden können, daß sie aber möglicherweise nicht alle in der bekannten Weise wirken, wie es etwa für das INPUT-Statement und auch den GET-Befehl gilt.

Kapitel 6: Editieren

6.1 Vorbemerkungen

Ein wesentlicher Teil der Zeit wird beim Programmieren aufgewendet für die Verbesserung von Fehlern - von Tippfehlern und Programmierfehlern - und die Einfügung von zusätzlichen Befehlen, die notwendig sind für einen korrekten Programmablauf oder auch nur ein Programm verbessern, weil sie z.B. Eingabefehler abfangen oder Bildschirmanzeigen übersichtlicher gestalten.

Das Neuschreiben von fehlerhaften Zeilen ist mühsam und birgt die Gefahr neuer Tippfehler. Spätestens wenn man dazu übergeht, mehrere Befehle in einer Zeile zusammenzufassen, wird es unsinnig, vielleicht wegen eines Zeichens alles neu zu tippen. Man kann es sich leichter machen, wenn auch der APPLE keinen fest implementierten Editor mit eigenem Befehlssatz hat. Eine Möglichkeit bietet die [ESC] -Taste, eine andere (komfortablere) der PROGRAM LINE EDITOR.

6.2 Der Escape-Modus

In diesem Modus haben einige Buchstabentasten eine andere Funktion als gewohnt. Wie schon aus den Handbüchern bekannt, sind dies die Tasten A bis F und I bis M.
Zur Erinnerung sei kurz erwähnt, daß die Bewegung des Cursors wie folgt geschieht:

Cursor 1 Zeile nach oben	-	I , D .
Cursor 1 Zeile nach links	-	J , B .
Cursor 1 Zeile nach rechts	-	K , A .
Cursor 1 Zeile nach unten	-	M , C .

Dabei hat man z.B. nach [ESC] [D] den Escape-Modus sofort wieder verlassen, nach [ESC] [I] jedoch nicht. Deshalb bietet sich die erste Spalte für die Cursorsteuerfunktionen eher an als die zweite. Man bleibt dann im Escape-Modus, solange man nur I,J,K oder M eingibt. Zudem kann man sich anhand der Lage der Tasten zueinander auf der Tastatur eher ihre Funktion einprägen.

Mit [ESC] [E] kann eine Bildschirmzeile von der aktuellen Cursorposition ab nach rechts gelöscht werden, mit [ESC] [F] wird der Bildschirm ab der aktuellen Position nach rechts und nach unten gelöscht.
In beiden Fällen wird der Escape-Modus verlassen.

Ebenso bekannt ist wohl auch die Tatsache, daß im Escape-Modus nur Bewegungen auf dem Bildschirm ausgeführt werden, aber weder Veränderungen des Bildschirm- noch des Speicherinhalts erfolgen.
Dies kann hingegen geschehen mit den "Pfeiltasten":

[←] (Backspace) und [→] (Retype).

Folgende Programmzeile soll als Beispiel für die Anwendung dieser Funktionen beim Editieren dienen:

```
100  PRINT : PRINT "DAS ERGRBNIS
     LAUTET";X: IF N = 5 THEN  END
```

Es sollen folgende Änderungen erfolgen:

1. Der Tippfehler in ERGEBNIS soll verbessert werden.
2. Hinter LAUTET soll innerhalb des Strings ein Doppelpunkt und ein Leerzeichen eingefügt werden.

Durch LIST 1ØØ bringt man zunächst einmal die Zeile auf den Bildschirm. Durch [ESC] [I] gelangt man in den Escape-Modus und um eine Zeile nach oben, weiteres Drücken von [I] bringt den Cursor in die richtige Zeile und [J] an den Zeilenanfang bis zur '1' von 1ØØ. Durch [→] wandert man nun über den Anfang der Zeile 1ØØ bis zum fehlerhaften 'R' in ERGRBNIS, dort wird der Fehler einfach überschrieben durch E und die Retype-Taste benutzt, um das restliche Wort einschließlich eines Leerzeichens neu einzulesen.

Der Rest der ersten Bildschirmzeile der Programmzeile 1ØØ ist leer, entsprechend der Bildschirmdarstellung des APPLE. Diese Leerzeichen "innerhalb eines Strings", der ja in der nächsten Zeile weitergeht, müssen im ESCAPE-Modus übersprungen werden. Durch [ESC] [K] gelangt man nach mehrfacher Anwendung zum 'L' von LAUTET, was ja wieder in die korrigierte Zeile aufgenommen werden soll.

Nach sechsmaligem Drücken der Retype-Taste steht der Cursor hinter dem Wort, auf den Anführungszeichen. Dort sollen nun 2 Zeichen eingefügt werden. Dies ist möglich durch folgendes Vorgehen:

[ESC] [M]	Verlassen der Zeile nach unten
[J] [J]	2 Zeichen nach links
Leertaste	Aufheben des Escape-Modus
[:] Leertaste	Eingabe der 2 neuen zusätzlichen Zeichen
[ESC] [I]	Rückkehr in die Programmzeile an die alte Stelle, d.h. bei dem Anführungszeichen

Durch wiederholtes Drücken der Retype-Taste kann nun noch der fehlende Rest der Zeile eingelesen werden und die neue Programmzeile wird durch [RETURN] abgeschlossen:

```
100  PRINT : PRINT "DAS ERGEBNIS
     LAUTET: ";X: IF N = 5 THEN  END
```

Eine wesentliche Erleichterung bei dieser Art des Editierens, insbesondere, wenn Strings in einer Zeile enthalten sind, bietet der Befehl

```
POKE 33,33
```

der schon aus Abschnitt 3.1 bekannt ist. Hiermit wird die Breite des Bildschirmfensters von 4Ø auf 33 reduziert, so daß keine überflüssigen Leerzeichen mehr auf dem Bildschirm erscheinen, wenn LIST 1ØØ ausgeführt wird. Vor dem POKE sollte ein HOME-Befehl abgesetzt werden, damit nicht die Reste alter Anzeigen außerhalb des Bildschirmfensters stehen bleiben.

Man kann den Escape-Modus auch verwenden, um Zeilen zusammenzufassen. Die bekannte Zeile 1ØØ könnte z.B. aus den folgenden zwei Zeilen entstanden sein:

```
100  PRINT : PRINT "DAS ERGEBNIS LAUTET: ";X
110  IF N = 5 THEN  END
```

Nach HOME und POKE 33,33 gibt man

```
LIST 1ØØ-11Ø
```

ein. Danach bringt man den Cursor durch [ESC] [I] über die '1' der Zeilennummer 1ØØ. Mit der Retype-Taste liest man den Inhalt erneut ein, bis der Cursor hinter dem 'X' steht.
Nun tippt man ':' ein. Durch [ESC] [M] und [J] wird der Cursor eine Zeile tiefer an den Anfang des Textes, also über das 'I' von IF positioniert und der Inhalt dieser Zeile mit Retype ebenfalls eingelesen.
Die Kontrolle durch LIST 1ØØ-11Ø ergibt:

```
100  PRINT : PRINT "DAS ERGEBNIS
     LAUTET: ";X: IF N = 5 THEN  END

110  IF N = 5 THEN  END
```

Nun kann Zeile 11Ø gelöscht werden. Auf ähnliche Weise kann man auch Zeilen wieder aufspalten, wenn längere Einschübe nötig werden, die eine solche Aufspaltung sinnvoll erscheinen lassen. Im obigen Beispiel wäre folgendermaßen vorzugehen:

```
     LIST 1ØØ
Eintippen der Zeilennummer 11Ø
Escape-Modus:  Cursor positionieren auf das 'I'
               von IF in Zeile 1ØØ
Retype-Taste   (bis hinter END)
 RETURN
```

Nun gibt es wieder eine Zeile 11Ø, die den IF... THEN-Befehl enthält. Dann kann Zeile 1ØØ entsprechend gekürzt werden:

LIST 1ØØ
Escape-Modus: Cursor positionieren auf die '1' von 1ØØ
Retype-Taste (bis hinter 'X')
RETURN

So werden ab dem Doppelpunkt alle Zeichen einfach weggeschnitten, da sie ja nicht erneut mit Retype eingelesen wurden.
Zeichen, die mitten aus einer Programmzeile gelöscht werden sollen, dürfen nicht mit Retype neu eingelesen werden, sondern werden mit [ESC] [K] "übersprungen".

Nachdem alle Editiervorgänge erledigt sind, sollte das Bildschirmfenster wieder auf seine normale Breite erweitert werden durch den Befehl

POKE 33,4Ø

Der Escape-Modus bietet also alle Möglichkeiten, schon eingegebene Programmzeilen nachträglich zu verändern, Zeilen zusammenzufassen oder aufzuspalten. Das Verfahren ist nicht so komfortabel wie ein Editor, aber schon nach kurzer Einübungsphase kann man es vernünftig und zeitsparend einsetzen.

6.3 PROGRAM LINE EDITOR

Das im folgenden kurz PLE oder Editor genannte Programm mit Copyright von Neil Konzen leistet eine bequeme Form der Veränderung, Kürzung oder Erweiterung von Programmzeilen. Das Zusatzprogramm ESC CREATE bietet darüber hinaus die Möglichkeit, häufig verwendete Befehle als Escapesequenzen abzurufen und einzelne Tasten für den Escape-Modus selbst neu zu belegen.
So kann etwa durch [ESC] [1] der Catalog der Diskette in Drive 1, durch [ESC] [2] derjenigen in Drive 2 abgerufen werden, d.h. statt 1o Tasten müssen nur noch 2 gedrückt werden, da z.B. ESC 1 gleichbedeutend ist mit CATALOGD1 [RETURN] . Diese Festlegungen können ergänzt oder abgeändert werden - je nach den Vorstellungen und Bedürfnissen des einzelnen Benutzers.
Diese Sonderfunktionen sind nur aktiviert, wenn der PLE geladen ist.
Dies geschieht einfach durch

RUN PROGRAM LINE EDITOR

Der Editor wird durch Herabsetzen von HIMEM vor dem Überschreiben geschützt, auch nach dem Befehl FP bleibt er erhalten, nach dem Booten durch PR# 6 ist er allerdings im Normalfall nicht mehr verfügbar, es sei denn, man hat ihn in alle Begrüßungsprogramme eingebaut. Dazu folgen nähere Erläuterungen in Abschnitt 7.3.

Durch CTRL-E wird der Editor aufgerufen. Es erscheint das Wort EDIT auf dem Schirm, dahinter ist die Nummer der gewünschten Zeile einzugeben, nach dem RETURN wird die Zeile auf dem Bildschirm angezeigt, der Cursor steht auf dem ersten Zeichen hinter der Zeilennummer, das kein Leerzeichen ist. Steuerzeichen werden invers sichtbar gemacht. In diesem Editiermodus haben einige Steuerzeichen besondere Funktionen, die die Zeile verändern, d.h. Zeichen löschen, einfügen, oder die Bewegung innerhalb der Zeile steuern.
Backspace und Retype funktionieren genau wie sonst. Falsche Zeichen können neu einfach durch Eingabe des richtigen Zeichens überschrieben werden.

Die wichtigsten Editorbefehle werden hier zunächst aufgelistet und dann wird ihr Einsatz an einem Beispiel demonstriert.

CTRL-I Insert - Übergang in den Einfügemodus

Vor der Stelle, an der der Cursor gerade steht, werden ein oder mehrere Zeichen eingefügt. Der rechts von dieser Stelle stehende Teil der Zeile wird weiter nach rechts gerückt.
Dieser Einfügemodus wird beendet durch Eingabe eines anderen Steuerzeichens, durch Retype oder RETURN .

CTRL-O Override - Übergang in den Einfügemodus

Es gilt das gleiche, was für die Einfügungen mit Hilfe von CTRL-I gesagt wurde; allerdings können nach CTRL-O auch Steuerzeichen in eine Programmzeile eingefügt werden. Erst nach Einfügen dieses Steuerzeichens (z.B. CTRL-D für Diskettenbefehle) ist der Einfügemodus beendet.

CTRL-D Delete

Es wird ein Zeichen an der Cursorposition gelöscht und der Text rechts davon um ein Zeichen weiter nach links gerückt.

CTRL-Z Zeichen Zap

Durch Eingabe von CTRL-Z und einem gewünschten Zeichen werden ab der Cursorposition alle Zeichen gelöscht bis zum ersten Auftreten des eingegebenen Zeichens. Dieses bleibt zunächst erhalten. Durch Wiederholung der Eingabe des gewünschten Zeichens kann der Löschvorgang ein zweites, drittes,... Mal durchgeführt werden. Das eingegebene Zeichen kann auch ein Steuerzeichen sein.

CTRL-P Pack

Mit diesem Befehl werden alle Leerzeichen einer BASIC-Zeile außerhalb von Strings gelöscht - ganz gleich, an welcher Stelle in der Zeile der Befehl abgesetzt wird - und der Cursor wird auf den Anfang der Zeile gesetzt.
Der Befehl wird nötig, wenn in sehr lange Zeilen noch Einfügungen gemacht werden sollen. Bei kurzen Zeilen hat CTRL-P nur vorübergehende Wirkung.

Die nächste Gruppe von Befehlen ermöglicht das Positionieren des Cursors innerhalb einer Zeile. Außer der Benutzung der "Pfeiltasten" Backspace und Retype gibt es im Editiermodus noch drei weitere Möglichkeiten:

CTRL-N End

Der Cursor wird hinter das letzte Zeichen der Zeile gesetzt.

CTRL-B Begin

Der Cursor wird auf das allererste Zeichen einer Zeile gesetzt, d.h. auf die erste Ziffer der Zeilennummer.

CTRL-F Zeichen Find

Der Cursor springt an die Stelle, an der das angegebene Zeichen zum ersten Mal auftaucht. Durch wiederholte Eingabe des Zeichens kann der Cursor weiter nach rechts bis zum nächsten Auftauchen des Zeichens positioniert werden.

Die dritte Gruppe wird von solchen Befehlen gebildet, mit denen man Änderungen einer Zeile abspeichern oder aufheben kann:

CTRL-R Restart

Durch diesen Befehl werden alle bisher durchgeführten Änderungen wieder aufgehoben und die editierte Zeile erscheint wieder in der ursprünglichen Form. Selbstverständlich gilt dies nur, wenn der Editiermodus noch nicht verlassen wurde. Nach CTRL-R befindet man sich immer noch im Editor, es können erneut Änderungen derselbe Zeile vorgenommen werden.
Die Cursorposition bei Eingabe des Befehls ist ohne Bedeutung.

CTRL-X Exit

Es werden alle bisher gemachten Änderungen aufgehoben und der Editor wird verlassen. Die gerade editierte Zeile bleibt in ihrer ursprünglichen Form im Arbeitsspeicher erhalten.
Es ist unerheblich, an welcher Stelle der Cursor in der Zeile gerade steht, wenn der Befehl abgesetzt wird.

CTRL-Q Quit

Bei diesem Befehl ist die Cursorposition wichtig.
Alle Änderungen, die in der Zeile links vom Cursor durchgeführt wurden, werden abgespeichert, alle Zeichen rechts davon, einschließlich des Zeichens an der aktuellen Cursorposition, werden gelöscht und der Editor wird verlassen.
Der Befehl eignet sich somit zum "Abschneiden" des rechten Teils von Zeilen.

CTRL-M RETURN

Alle Änderungen der Zeile werden gespeichert und der Editormodus wird verlassen, die Cursorposition bei Eingabe des Befehls ist unerheblich.
Statt CTRL-M kann natürlich auch die RETURN-Taste gedrückt werden.

Mit diesen Befehlen sind alle erwünschten Änderungen von Programmzeilen auf komfortable Weise durchzuführen. Darüber hinaus gibt es noch Befehle, die Cursorsteuerfunktion haben und in PRINT-Befehle eingefügt werden können, so daß der Cursor programmgesteuert auf dem Bildschirm in alle 4 Richtungen bewegt werden kann.

Außerdem ist es möglich, im Editiermodus von Groß- auf Kleinschreibung umzuschalten durch einen entsprechenden Befehl (CTRL-S). Falls der APPLE nicht mit einem entsprechenden Zeichengenerator versehen ist, erscheinen auf dem Bildschirm keine Kleinbuchstaben, sondern unverständliche Folgen von Symbolen. Auf einem entsprechenden Drucker werden aber die kleinen Buchstaben ausgegeben.

An einem Beispiel sollen zum Abschluß einige Verwendungsmöglichkeiten der Editorbefehle aufgezeigt werden; es handelt sich um ein kleines Programm, das die Benzinkosten pro Person in einer Fahrgemeinschaft berechnet:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 143.9: REM  BENZINPREIS IN PF
40  INPUT KM
50  INPUT N
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 40
70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
110  PRINT : PRINT "STRECKE IN KM      : ";KM
120  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
130  END
```

Nun soll zunächst in Zeile 3Ø der Wert für P verändert werden, da der Benzinpreis gesunken ist:

CTRL-E	Es erscheint die Anzeige: EDIT -
3Ø RETURN	Die Zeile wird sichtbar, der Cursor steht auf dem "P".
→ (5 mal)	Der Cursor steht auf der "4".
3	Die Ziffer "4" wird durch "3" überschrieben.
6	Die "3" wird durch "6" überschrieben.
RETURN	Der Editor wird verlassen, die Änderung ist gespeichert:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
40  INPUT KM
50  INPUT N
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 40
70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
110  PRINT : PRINT "STRECKE IN KM    : ";KM
120  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
130  END
```

In der Zeile 4Ø soll das INPUT-Statement durch eine Erklärung erweitert werden, daß eine Eingabe für die Fahrtstrecke erwartet wird:

CTRL-E 4Ø RETURN	Aufruf des Editors, der Cursor steht auf dem "I".
CTRL-F K	Der Cursor spring zum "K" von KM.
CTRL-I	Übergang in den Einfügemodus.
"FAHRTSTRECKE: ";	Eingabe des neuen Teiles.
RETURN	Abschluß des Einfügemodus und der Änderungen:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
→40  INPUT "FAHRTSTRECKE: ";KM
50  INPUT N
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 40
70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
110  PRINT : PRINT "STRECKE IN KM     : ";KM
120  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
130  END
```

In Zeile 5Ø soll ebenfalls eine Erläuterung dem INPUT hinzugefügt werden, das Vorgehen ist also ähnlich:

CTRL-E 5Ø RETURN	Nach Aufruf des Editors steht der Cursor auf dem "I".
→ (6 mal)	Der Cursor steht auf N, dem Zeichen, vor dem die Einfügung erfolgen soll.
CTRL-I	Einfügemodus
"PERSONENZAHL: ";	Eingabe des neuen Textes
CTRL-Q	Versehentlich wird der Editor durch CTRL-Q verlassen, was ein Abschneiden des Variablennamens N zur Folge hat:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
40  INPUT "FAHRTSTRECKE: ";KM
→50  INPUT "PERSONENZAHL: ";
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 40
70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
110  PRINT : PRINT "STRECKE IN KM    : ";KM
120  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
130  END
```

Die Zeile 5Ø muß wieder vervollständigt werden:

CTRL-E 5Ø RETURN	
CTRL-N	Der Cursor springt hinter das "; "
N	Das "N" wird an die bisherige Zeile angehängt.
RETURN	

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
40  INPUT "FAHRTSTRECKE: ";KM
50  INPUT "PERSONENZAHL: ";N
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 40
70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
110  PRINT : PRINT "STRECKE IN KM     : ";KM
120  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
130  END
```

In Zeile 6Ø steht eine falsche Sprungadresse, die Zeilennummer 4Ø muß durch 5Ø ersetzt werden:

CTRL-E 6Ø RETURN

CTRL-F 4 Der Cursor überspringt alle Zeichen bis zur falsch eingegebenen 4.

5 Die richtige Ziffer wird eingegeben.

RETURN

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
40  INPUT "FAHRTSTRECKE: ";KM
50  INPUT "PERSONENZAHL: ";N
→60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 50
70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
110  PRINT : PRINT "STRECKE IN KM     : ";KM
120  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
130  END
```

Der Programmierer denkt nun irrtümlich, die Division durch 1ØØ in Zeile 7Ø sei ein Fehler gewesen.
Er will die 1ØØ löschen:

CTRL-E 7Ø RETURN	
→ (1ø mal)	Der Cursor steht auf der "1"
CTRL-D	Löschen der "1"
CTRL-D	Löschen der "Ø"
CTRL-D	Löschen der "Ø"

Nun sieht das Programm so aus:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
40  INPUT "FAHRTSTRECKE: ";KM
50  INPUT "PERSONENZAHL: ";N
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 50
70 GV = KM /  * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
110  PRINT : PRINT "STRECKE IN KM     : ";KM
120  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
130  END
```

Aber an dieser Stelle fällt ihm auf, daß er sich geirrt hat; die Zeile ist völlig korrekt:

CTRL-X Der Editor wird verlassen, die Änderungen werden wieder aufgehoben. Die Zeile bleibt in der ursprünglichen Version im Programm.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
40  INPUT "FAHRTSTRECKE: ";KM
50  INPUT "PERSONENZAHL: ";N
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 50
→70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
110  PRINT : PRINT "STRECKE IN KM     : ";KM
120  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
130  END
```

Jetzt soll der Ergebnisausdruck erweitert werden.
Als erstes soll hinter HOME eine Zeile eingefügt werden, die die Anzeige der Personenzahl bewirkt.
Außerdem soll aber die Numerierung der Zeilen in Zehnerschritten beibehalten werden, d.h. die Zeilen 11Ø-13Ø müssen in 12Ø-14Ø umbenannt werden:

CTRL-E 13Ø RETURN

CTRL-B Der Cursor springt auf die erste Ziffer der Zeilennummer.

→ 4 Die 3 wird in 4 geändert.

RETURN

```
CTRL-E 12Ø   RETURN

CTRL-B
 ⟶ 3          2 wird zu 3

  RETURN


CTRL-E 11Ø   RETURN

CTRL-B
 ⟶ 2          1 wird 2

  RETURN
```

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
40  INPUT "FAHRTSTRECKE: ";KM
50  INPUT "PERSONENZAHL: ";N
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 50
70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
→110  PRINT : PRINT "STRECKE IN KM     : ";KM
120  PRINT : PRINT "STRECKE IN KM     : ";KM
130  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
140  END
```

Die Zeilen 11Ø und 12Ø sind nun identisch; Zeile 11Ø soll den neuen Text bekommen. Deshalb wird zunächst mit dem Löschen des alten begonnen:

CTRL-E 11Ø RETURN

CTRL-D (7 mal)

Nach diesem Löschen fällt auf, daß eine Änderung des Strings viel günstiger ist. Das Programm sieht jetzt so aus:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
40  INPUT "FAHRTSTRECKE: ";KM
50  INPUT "PERSONENZAHL: ";N
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 50
70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
110  PRINT "STRECKE IN KM     : ";KM
120  PRINT : PRINT "STRECKE IN KM     : ";KM
130  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
140  END
```

Da der Irrtum vor dem Verlassen des Editors bemerkt wurde, kann der ursprüngliche Zustand der Zeile leicht wiederhergestellt werden:

CTRL-R	Die bisherigen Änderungen sind aufgehoben, aber die Zeile kann weiter editiert werden.
CTRL-F S	Der Cursor wird auf "S" von "STRECKE" gesetzt.
"PERSONENZAHL "	Der alte String wird durch den neuen überschrieben.

RETURN

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  PROGRAMM ZUR BERECHNUNG DER FAHRTKOSTEN IM PKW
BEI VARIABLER PERSONENZAHL UND VARIABLER FAHRTSTRECKE
15  REM  BENZINPREIS UND VERBRAUCH WERDEN ALS FEST ANGESEHEN
20 V = 10: REM  VERBRAUCH IN L AUF 100 KM
30 P = 136.9: REM   BENZINPREIS IN PF
40  INPUT "FAHRTSTRECKE: ";KM
50  INPUT "PERSONENZAHL: ";N
60  PRINT : IF N > 5 THEN  PRINT "MEHR ALS 5 PERSONEN IM
PKW IST VERBOTEN!": PRINT : GOTO 50
70 GV = KM / 100 * V
80 K = GV * P / 100
90 KP = K / N:KP =  INT (KP * 100 + .5) / 100
100  HOME
→110  PRINT : PRINT "PERSONENZAHL      : ";KM
120  PRINT : PRINT "STRECKE IN KM      : ";KM
130  PRINT : PRINT "KOSTEN PRO PERSON: ";KP
140  END
```

Damit soll das Programm seine endgültige Form erhalten haben, weitere Änderungen sind nicht erwünscht. An diesem Beispiel konnten zwar nicht alle Befehle demonstriert werden, doch wird wohl auch so schon deutlich, wie bequem Änderungen in Programmzeilen durchgeführt werden können mit Hilfe des Programms PROGRAM LINE EDITOR.
Das Zusammenfassen von Zeilen muß nach wie vor im Escape-Modus erfolgen, doch kann auf POKE 33,33 verzichtet werden, wenn die betreffenden Zeilen nicht mit LIST, sondern mit CTRL-E auf dem Bildschirm angezeigt werden.

Es wurde schon erwähnt, daß nach dem Start des PLE einige ESCAPE-Sequenzen eine besondere Bedeutung haben. So bewirkt beispielsweise [ESC] [:] den Aufruf des Monitors. Will man also den Doppelpunkt zwischen die beiden zusammenzufassenden Zeilen einfügen, muß sichergestellt sein, daß der Escape-Modus bereits verlassen wurde. Man kann aber auch die ESC-Funktionen neu belegen.

Ein Nachteil des PLE ist die etwas umständliche Art des Zeilenumnumerierens. Hier bringt die Verwendung des Programms nur insofern einen Vorteil, als nicht mehr die ganze Zeile durch Retype neu eingelesen werden muß. Im nächsten Kapitel wird allerdings ein Hilfsprogramm vorgestellt, das in diesem Bereich wesentlich komfortabler ist.

Kapitel 7: Programmierhilfen

7.1 Vorbemerkungen

Unter dem Oberbegriff "Programmierhilfen" sollen in diesem Kapitel mehrere Techniken angesprochen werden, die ein Programm schneller oder benutzerfreundlicher machen, bzw. solche Hilfsroutinen vorgestellt werden, die die Programmierarbeit erleichtern.

In den ersten beiden Abschnitten werden einige Ersatzmöglichkeiten der APPLE-Programmpakete APPLE DOC und DOS TOOL KIT vorgeführt, darunter besonders die Routinen APA (APPLE Programmer's Assistant) und LINEDOC und VARDOC, die fast unentbehrliche Hilfsmittel sind für den fortgeschrittenen APPLE-Benutzer, der schon umfangreichere Programme schreibt, bei denen man den Überblick über die verwendeten Variablen und die Sprungadressen leicht verlieren kann.

Im dritten Abschnitt wird an einem Beispiel demonstriert, wie man sein Initialisierungsprogramm so erweitern kann, daß bei jedem Einschalten des Rechners die häufig benutzten Routinen sofort zur Verfügung stehen.

Die beiden letzten Abschnitte schließlich zeigen fortgeschrittene Programmierung an zwei Spezialfällen auf, die sich dem Programmierer besonders häufig als Problem stellen: Sortieren und Bildschirmgestaltung.

Beim Sortieren zeigt sich besonders deutlich, wie die Wahl eines geeigneten Algorithmus, d.h. die Wahl des geeigneten Lösungswegs, die Ausführungszeit eines Programms erheblich verkürzen kann.

Die Gestaltung guter Bildschirmmasken - das sind solche, die übersichtlich und für den Benutzer leicht verständlich sind - ist eine zeitintensive Aufgabe für den Programmierer.
Durch Anwendung kleiner Tricks kann man dafür schon bald erheblich Zeit einsparen.

7.2 Umnumerieren und Mischen von Programmen

Das DOS TOOL KIT (tool kit = Werkzeugkasten) - mit Copyright der Firma APPLE Computer Incorporation - enthält neben vielen anderen Programmen (Spiele, verschiedene Zeichensätze) APA, ein nützliches Hilfsmittel bei der Erstellung längerer Programme, mit dem u.a. ganze Programme oder Programmteile umnumeriert oder mehrere Programme "gemischt" werden können.
Auf der Diskette befinden sich die Programme APA und LOADAPA. Zunächst muß mit

```
RUN LOADAPA
```

das Ladeprogramm gestartet werden, das APA genau unterhalb von HIMEM lädt und dann HIMEM herabsetzt, so daß unser Hilfsprogramm gegen versehentliches Löschen geschützt ist. Das Herabsetzen von HIMEM "unterhalb" von APA bedeutet Schutz für APA und zugleich eröffnet es die Möglichkeit, nun im folgenden "ganz normal" mit dem Rechner zu arbeiten. Es können Programme geladen oder eingetippt werden, als sei APA nicht vorhanden. Eine Veränderung ergibt sich natürlich: Der frei zur Verfügung stehende Speicher wird durch ein niedrigeres HIMEM kleiner. Nach dem Booten durch die Systemdiskette steht HIMEM auf 38400, nach dem Laden von APA ist die höchste ansprechbare Speicheradresse 34785.
Dieser Verlust von 3615 bytes wird sich jedoch erst bei sehr großen Programmen mit sehr vielen Variablen bemerkbar machen.

LOADAPA liefert außerdem folgenden Bildschirmausdruck, eine Übersicht über die möglichen APA-Befehle:

```
APPLESOFT PROGRAMMER'S ASSISTANT
VERSION 1.0
(C) COPYRIGHT 1979, APPLE COMPUTER INC.

&RENUMBER <START>,<INC>,<FIRST>,<LAST>
&HOLD
&MERGE
&LENGTH
&COMPRESS
&SHOW
&NOSHOW
&AUTO <START>,<INC>
&MANUAL
&XREF
&KEYS
```

Wie schon aus der Übersicht deutlich wird, müssen alle Befehle des APA durch "&" gekennzeichnet werden zur Unterscheidung von den üblichen Kommandos, d.h. Befehlen im Direktmodus. Von besonderem Interesse sind hier die Befehle &RENUMBER, &HOLD und &MERGE, deren Wirkung ausführlich besprochen werden soll.

Was leisten die übrigen?

&LENGTH

Die Länge eines im Arbeitsspeicher befindlichen Programms kann ermittelt werden. Nach dem Eintippen oder Laden eines Programms muß über die Tastatur nur

&L

eingegeben werden und die Länge des Programms wird in bytes angezeigt, und zwar dezimal und hexadezimal.

&COMPRESS

Aus dem Programm im Arbeitspeicher werden alle REM-Statements entfernt nach Eintippen von

&C

Es erfolgt eine Anzeige, wieviele bytes eingespart werden konnten, und zwar wiederum dezimal und hexadezimal.

&SHOW

Nach Eingabe des entsprechenden Befehls

&S

sind zunächst unsichtbare Steuerzeichen (control characters) im Programm als inverse Zeichen sichtbar. Der Befehl bleibt in Kraft, bis er aufgehoben wird durch den nachfolgenden.

&NOSHOW

&N

hebt den Befehl &S wieder auf, die Steuerzeichen werden wieder unsichtbar.

&AUTO ⟨start⟩ , ⟨inc⟩

Dieser Befehl ermöglicht eine automatische Zeilennumerierung, die bei der angegebenen Startzeile beginnt und jeweils um die Schrittweite (increment) erhöht. Beide Parameter müssen angegeben werden. Durch Eingeben von z.B.

&A 1ØØ,1Ø

wird die automatische Numerierung initialisiert, und zwar wird als erstes die Zeilennummer 1ØØ generiert, die nachfolgenden jeweils in Zehnerschritten: 11Ø, 12Ø, 13Ø.
Dies geschieht aber nur, wenn als erstes Zeichen nach einem <RETURN> <SPACE>(Leerzeichen) über die Tastatur abgeschickt wird.
Es bedeutet aber, daß nach wie vor Kommandos wie RUN, LOAD, SAVE etc. abgesetzt werden können.
Außerdem kann auch eine Numerierung "von Hand" weiterhin erfolgen, wenn eine von <INC> abweichende Schrittweite gewünscht wird. Wenn im angegebenen Beispiel aber die Zeilennummer 12Ø nicht durch Drükken der Leertaste, sondern durch Eintippen der Zahlen generiert wird, so wird dies von APA nicht registriert, d.h. diese Nummer wird von der Automatik noch zusätzlich erzeugt. Ebenso wird nicht geprüft, ob hinter einer automatisch erzeugten Zeilennummer tatsächlich etwas eingegeben wurde.
Wenn man sofort die <RETURN> -Taste drückt, wird diese Zeile zwar wie üblich beim Listing nicht aufgeführt, aber APA erhöht trotzdem entsprechend der angegebenen Schrittweite. Nur wenn <u>vor</u> dem Abschluß der Zeile, d.h. vor dem <RETURN> , CTRL-X eingegeben wird, wird die zuletzt erfolgte Zeilennummererhöhung rückgängig gemacht. Dies kann benutzt werden, um aus falsch begonnenen Zeilen wieder herauszukommen.

Die Schrittweite kann durch weitere Befehle verändert werden, auch nachdem Teile eines Programms eingegeben worden sind. Aufgehoben wird dieser Befehl durch den nachfolgenden.

&MANUAL

Durch Eintippen von

&MA

wird die automatische Zeilennumerierung wieder "ausgeschaltet".

&XPRET

Dieser Befehl erstellt eine alphabetisch sortierte Liste aller Variablen, die in dem im Arbeitsspeicher vorhandenen Programm verwendet wurden, mit den Nummern der Zeilen, in denen die jeweilige Variable auftauchte. Dabei werden Arrayvariablen durch eine nachfolgende linke Klammer gekennzeichnet, Stringvariablen und Integervariablen wie üblich durch $ und %. Genau wie bei Programmlistings kann die Bildschirmanzeige durch CTRL-S unterbrochen werden. Einzutippen ist nur

&X

Im nächsten Abschnitt wird eine weitere Möglichkeit für eine Variablentabelle vorgestellt.

&KEYS

Durch diesen Befehl werden 3 zusätzliche Zeichen verfügbar, die dann über Steuerzeichen abrufbar sind. Nach Eintippen von

&K

ist die linke eckige Klammer ([ ) darstellbar durch CTRL-K, der Backslash (\) durch CTRL-L und der Strich auf der Grundlinie (underscore: _) durch CTRL-O.

Aufgehoben wird dieser Befehl ebenfalls durch

&MA

Bevor nun die 3 wichtigsten Befehle des APA vorgestellt werden, soll noch kurz darauf hingewiesen werden, daß einige dieser Befehle, beispielsweise &AUTO, andere Hilfsroutinen zerstören, die möglicherweise vorher geladen worden waren. Dieses Problem wird in Abschnitt 7.4 noch einmal aufgegriffen.

Nun aber zu den 3 Befehlen, deren Syntax zunächst erläutert werden soll:

&HOLD

Mit diesem Befehl wird das Programm im Arbeitsspeicher "beiseitegeschoben"; es kann ein neues geladen werden, aber das alte geht nicht verloren.

&MERGE

Damit können 2 Programme gemischt werden durch folgende Vorgehensweise:
Ein erstes Programm wird geladen und durch

&H

zur Verfügung gehalten. Dann wird ein zweites Programm geladen, das andere Zeilennummern als das erste haben sollte. Durch

&M

werden die beiden Programme gemischt; ihre Zeilen werden nach den jeweiligen Nummern ineinandersortiert. Treten doch doppelte Zeilennummern auf, macht das Programm eine entsprechende Meldung und fragt

CONTINUE?

Bei der Antwort "N" wird die Ausführung des MERGE-Befehls abgebrochen, bei der Antwort "Y" wird gegebenenfalls die Zeile aus dem HOLD-Programm übernommen und die andere gelöscht.

&RENUMBER ⟨start⟩ , ⟨inc⟩ , ⟨first⟩ , ⟨last⟩

Mit diesem Befehl kann die Zeilennumerierung des ganzen Programms oder von Teilen des Programms im Arbeitsspeicher verändert werden, solange durch teilweise Umnumerierung keine überlappenden oder doppelten Zeilennummern entstehen. In solchen Fällen wird der Befehl nicht ausgeführt und es erfolgt eine entsprechende Bildschirmanzeige.
Alle 4 Parameter des Befehls sind optional, d.h. sie können weggelassen werden, das Programm benötigt aber die Kommata, da die Reihenfolge der Eingabe wichtig ist. Falls eine Parameterangabe fehlt, werden die Default-Werte angenommen. Voreingestellt sind folgende Werte:

⟨start⟩:	1ØØ	Für die erste neue Zeilennummer ist 1ØØ vorgesehen.
⟨inc⟩ :	1Ø	Der Abstand zwischen zwei aufeinanderfolgende Zeilennummern (d.h. die Schrittweite) wird mit 1Ø angenommen.

⟨first⟩ :	Ø	Die erste neu zu numerierende Zeile hat die kleinste mögliche Zeilennummer Ø, d.h. die Änderung beginnt bei der ersten Zeile des Programms.
⟨last⟩ :	63999	Dies ist die größtmögliche Zeilennummer, d.h. die Änderung wird bis zur letzten Zeile des Programms durchgeführt.

An einem Beispiel sollen nun einige Formen des Befehls vorgeführt werden. Im Arbeitsspeicher befindet sich das folgende Programm:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM     DEMONSTRATION VON
11  REM          R E N U M B E R
12  REM
20  PRINT "R"
30  PRINT "E"
40  PRINT "N"
50  PRINT "U"
60  PRINT "M"
70  PRINT "B"
80  PRINT "E"
90  PRINT "R"
100  END
```

Das Eintippen von

```
&R 1ØØ,,1Ø
```

führt zu folgender Version:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
100  REM     DEMONSTRATION VON
110  REM         R E N U M B E R
120  REM
130  PRINT "R"
140  PRINT "E"
150  PRINT "N"
160  PRINT "U"
170  PRINT "M"
180  PRINT "B"
190  PRINT "E"
200  PRINT "R"
210  END
```

Um hieraus die Ursprungsversion wiederherzustellen, muß man in 2 Schritten vorgehen

```
&R 1Ø,1,1ØØ,12Ø
```

Die erste neue Zeilennummer ist die 1Ø, die Schrittweite soll 1 sein und dies geschieht von Zeile 1ØØ bis zur Zeile 12Ø.

Als Ergebnis dieses Befehls erhalten wir folgende Programmversion:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM     DEMONSTRATION VON
11  REM          R E N U M B E R
12  REM
130  PRINT "R"
140  PRINT "E"
150  PRINT "N"
160  PRINT "U"
170  PRINT "M"
180  PRINT "B"
190  PRINT "E"
200  PRINT "R"
210  END
```

Folgender Renumber-Befehl stellt nun die ursprüngliche Programmnumerierung wieder her:

```
&R 2Ø,,13Ø
```

Die erste neue Zeile wird die Zeile 2Ø, die Schrittweite ist weggelassen worden, wird also auf den Defaultwert 1Ø gesetzt, die Umnumerierung beginnt bei der Zeile 13Ø des Programms und läuft bis zur letzten Zeile, da auch der Wert für last nicht spezifiziert ist.

Nun befindet sich wieder die erste Version im Arbeitsspeicher. Der nachfolgende Befehl wird von APA nicht ausgeführt, stattdessen bringt er die Fehlermeldung

INTERLEAVED OR DUPLICATE LINE NUMBER:

&R 36,6,7Ø,9Ø

Der Rechner wird angewiesen, die Zeilen 7Ø, 8Ø und 9Ø des Programms, die die Anzeige von B, E und R bewirken, umzunumerieren mit Schrittweise 6 und beginnend bei 36, d.h. er sollte folgende Zeilennummern erzeugen:+

```
3   HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4   PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM     DEMONSTRATION VON
11  REM         R E N U M B E R
12  REM
20  PRINT "R"
30  PRINT "E"
36  PRINT "B"
40  PRINT "N"
42  PRINT "E"
48  PRINT "R"
50  PRINT "U"
60  PRINT "M"
100  END
```

+Diese Numerierung wurde nicht mit APA erstellt, sondern mit dem RENUMBER von der APPLE Systemdiskette.

Man sieht, daß sich neue und alte Zeilennummern "überlappen", da die neue Zeile 36 zwischen 3ø und 4ø einsortiert wird, die beiden anderen (42 und 48) aber hinter die Zeile 4ø. Solche Überschneidungen werden nicht ausgeführt. Es ist aber möglich, zwischen zwei vorhandene Zeilen eine oder mehrere zwischenzuschieben, z.B. durch folgenden Befehl :

```
&R 53,3,7ø,9ø
```

Man erhält nun diese Numerierung :

```
3   HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4   PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM     DEMONSTRATION VON
11  REM          R E N U M B E R
12  REM
20  PRINT "R"
30  PRINT "E"
40  PRINT "N"
50  PRINT "U"
53  PRINT "B"
56  PRINT "E"
59  PRINT "R"
60  PRINT "M"
100  END
```

Solche Verschiebungen im Programm werden nötig, wenn der Programmablauf nicht korrekt bzw. nicht optimal gestaltet ist. Eventuell müssen vor dem "Einschub" von Zeilen zwischen zwei andere größere Teile des Programms umnumeriert werden; dies wird immer dann erforderlich, wenn die Schrittweite zwischen den beiden "flankierenden" Zeilen nicht mindestens um Eins größer ist als die Anzahl der einzuschiebenden Zeilen. Nach der Veränderung des Programmablaufs kann durch ein weiteres Renumber wieder die ansonsten verwendete Schrittweite hergestellt werden.

Nicht ausgeführt würde im Gegensatz zu dem vorigen der folgende Befehl, dessen Ausführung zu doppelten Zeilennummern führen würde:

```
&R 1Ø,1Ø,1Ø,12
```

Die Zeilen 1Ø, 11 und 12 müßten umgewandelt werden zu 1Ø, 2Ø und 3Ø; dies bedeutete aber eine doppelte Verwendung der Zeilennummern 2Ø und 3Ø.
Die Ausführung unterbleibt und stattdessen erscheint auf dem Bildschirm die schon bekannte Meldung

```
INTERLEAVED OR DUPLICATE LINE NUMBER.
```

Diese Liste von Möglichkeiten macht einige Vorzüge der Hilfsroutine RENUMBER deutlich, doch fehlt noch der entscheidende Hinweis auf eine Eigenschaft, die anhand des Beispiels nicht sichtbar wurde.

Die Umnumerierungen beschränken sich nicht nur auf die eine Zeile anführende Zeilennummer, sondern auch auf die Sprungadressen, die in den Anweisungen GOTO, IF...THEN, GOSUB verwendet werden.
Für jede geänderte Zeilennummer wird überprüft, ob sie in einem Sprungbefehl genannt ist und auch dort wird sie dann geändert. Diese Überprüfung erstreckt sich auf das ganze Programm, auch dann, wenn nur Teile des Programms umnumeriert werden.

Dieser Abschnitt soll nun beendet werden mit einem Beispiel, das den großen Nutzen von APA bei der praktischen Programmierarbeit deutlich macht. Obwohl die APPLE-Systemdiskette ebenfalls ein RENUMBER-Programm enthält, ist APA ein Gewinn, da dieses Programm zuverlässiger arbeitet und z.B. mit dem Editor, der in Kapitel 6 beschrieben wurde, weitgehend verträglich ist, zumindest was die 3 wichtigsten Befehle angeht, die zuletzt besprochen wurden.

Beim einfachen RENUMBER treten hingegen Interaktionen mit dem Editor auf, die z.B. dazu führen, daß nach einem RENUMBER-Befehl der Monitor aufgerufen wird. Zwar ist der Befehl meist durchgeführt worden, aber es ist doch recht lästig, für das weitere Arbeiten am Programm immer erst 3DØG eintippen zu müssen, um den Monitor wieder zu verlassen.

Nun aber zum Beispiel:

Das folgende kleine Programm mit den Zeilennummern 1Ø-5Ø soll als Unterprogramm in ein Hauptprogramm eingefügt werden. Es erzeugt eine "laufende Bildschirmzeile". Die Variablen AØ und H1 müssen aus dem Hauptprogramm übergeben werden.

```
10 H$ = "                                        ":A =  LEN
(A$):A$ = A$ +  RIGHT$ (H$,40 - A)
20 H$ =  LEFT$ (A$,1):A$ =  RIGHT$ (A$,39) + H$: INVERSE : VTAB
(H1): CALL  - 958
30  VTAB (H1): PRINT  LEFT$ (A$,40): FOR Q = 1 TO 150: NEXT

40  IF  PEEK ( - 16384) < 128 THEN 20
50  NORMAL
```

Außerdem soll das fertig gemischte Programm mit dem bekannten Vorspann versehen werden. Dies geschieht durch das nachstehende kleine Programm aus den Zeilen 3Ø und 4Ø:

```
30  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
40  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
```

Das Hauptprogramm diene der Berechnung des Durchschnitts von N Werten. Dabei ist dem Programmierer ein Fehler unterlaufen, den er aber rechtzeitig bemerkt:

```
10  HTAB 7: PRINT "BERECHNUNG DES DURCHSCHNITTS"
20 A$ = "DRUECKEN SIE IRGENDEINE TASTE":H1 = 20: GOSUB 100
30  INPUT "WERT";X
40  INPUT "ANZAHL DER WERTE";N
50 S = S + X:I = I + 1: IF I < N THEN 30
60  HOME : VTAB 10: PRINT "DURCHSCHNITT DER ";N;" WERTE: ";S
/ N
70  END
```

Es handelt sich natürlich um die Vertauschung der beiden Zeilen 3Ø und 4Ø. Vor der Zeile 1Ø soll nun das Vorspannprogramm gesetzt werden, ab Zeile 1ØØ soll als Unterprogramm die laufende Bildschirmzeile folgen, die den Ausdruck

"DRUECKEN SIE IRGENDEINE TASTE"

in der Bildschirmzeile 2Ø produzieren soll.
Auf der Diskette befinden sich die Programme LAUFZEILE und VORSPANN, im Arbeitsspeicher das Programm zur Durchschnittsberechnung mit dem genannten Fehler.

Wie ist nun vorzugehen?

1. &R 25,,4Ø,4Ø

   Durch diesen Befehl wird die Zeile 4Ø zur Zeile 25 gemacht. Das Hauptprogramm ist jetzt im Ablauf korrekt:

```
10  HTAB 7: PRINT "BERECHNUNG DES DURCHSCHNITTS"
20 A$ = "DRUECKEN SIE IRGENDEINE TASTE":H1 = 20: GOSUB 100
25  INPUT "ANZAHL DER WERTE";N
30  INPUT "WERT";X
50 S = S + X:I = I + 1: IF I < N THEN 30
60  HOME : VTAB 10: PRINT "DURCHSCHNITT DER ";N;" WERTE: ";S
/ N
70  END
```

&H

Das Hauptprogramm wird beiseite geschoben.

2. LOAD VORSPANN

Das erste einzumischende Programm wird geladen. Es hat die Zeilennummern 3Ø und 4Ø, die zunächst geändert werden sollen in 3 und 4 durch:

&R 3,1

Ist das geschehen, werden diese beiden neuen Zeilen dem Hauptprogramm hinzugefügt durch

&M

Ergebnis dieser Befehle ist das folgende Programm:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HTAB 7: PRINT "BERECHNUNG DES DURCHSCHNITTS"
20 A$ = "DRUECKEN SIE IRGENDEINE TASTE":H1 = 20: GOSUB 100
25  INPUT "ANZAHL DER WERTE";N
30  INPUT "WERT";X
50 S = S + X:I = I + 1: IF I < N THEN 30
60  HOME : VTAB 10: PRINT "DURCHSCHNITT DER ";N;" WERTE: ";S
/ N
70  END
```

Durch &H wird dieses wiederum beiseitegeschoben.

3. Nun fehlt noch das Unterprogramm. Es wird geladen durch

LOAD LAUFZEILE

Als erstes wird das noch fehlende RETURN in Zeile 6Ø hinzugefügt, da es ja ein Unterprogramm werden soll, und vorher wird der Bildschirm gelöscht, also:

6Ø HOME:RETURN

Nun muß es umnumeriert werden, da im Hauptprogramm der Befehl GOSUB 1ØØ steht.

&R

reicht ohne weitere Spezifikation aus, um die Zeilen 1Ø bis 6Ø umzuwandeln in die Zeilen 1ØØ bis 15Ø.

Man beachte, daß auch die Sprungadresse in der ehemaligen Zeile 4Ø (jetzt 13Ø) geändert wurde in 11Ø.

Das Einmischen in das bisher erstellte Hauptprogramm geschieht durch

```
&M
```

wie schon vorher. Nun wird das Hauptprogramm noch umnumeriert, um die ungewohnte Zeilennummer 25 zu vermeiden, nämlich durch

```
&R 3Ø,,25
```

Das Ergebnis ist nun das fertige Programm:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HTAB 7: PRINT "BERECHNUNG DES DURCHSCHNITTS"
20 A$ = "DRUECKEN SIE IRGENDEINE TASTE":H1 = 20: GOSUB 100
30  INPUT "ANZAHL DER WERTE";N
40  INPUT "WERT";X
50 S = S + X:I = I + 1: IF I < N THEN 40
60  HOME : VTAB 10: PRINT "DURCHSCHNITT DER ";N;" WERTE: ";S
/ N
70  END
100 H$ = "                                        ":A =  LEN
(A$):A$ = A$ +  RIGHT$ (H$,40 - A)
110 H$ =  LEFT$ (A$,1):A$ =  RIGHT$ (A$,39) + H$: INVERSE :
VTAB (H1): CALL  - 958
120  VTAB (H1): PRINT  LEFT$ (A$,40): FOR Q = 1 TO 150: NEXT

130  IF  PEEK ( - 16384) < 128 THEN 110
140  NORMAL
150  HOME : RETURN
```

Auch bei diesem Umnumerieren wurde eine Sprungadresse verändert.
Falls nun noch mehr als zwei Zeilen in das Hauptprogramm eingefügt werden sollten, man aber nach wie vor in Zehnerschritten numerieren möchte, so müßte das Unterprogramm größere Zeilennummern bekommen, etwa mit Zeile 2ØØ beginnend. Der Leser überzeuge sich, daß der Befehl

```
&R 2ØØ,,1ØØ
```

zu einer Umnumerierung führt, die sowohl die Sprungadresse innerhalb des Unterprogramms von 11Ø in 21Ø umwandelt auch auch die Sprungadresse hinter GOSUB im Hauptprogramm auf 2ØØ setzt.

Zum Abschluß soll noch darauf hingewiesen werden, daß die beiden Hilfsroutinen APA und PROGRAM LINE EDITOR nur dann weitgehend miteinander verträglich sind, wenn zunächst der Editor und dann erst APA geladen werden. Bei der umgekehrten Reihenfolge wird APA durch den Editor zerstört.

Die Benutzung von Textfiles und des EXEC-Befehls, wie sie in Kapitel 5 beschrieben wurde, zum Mischen von fertigen Programmteilen ist bei weitem nicht so komfortabel wie der Einsatz von APA, da dessen Möglichkeiten vielseitiger sind.

7.3 Überblick und Dokumentation

Beim Mischen von Programmen oder auch beim Programmieren eines umfangreichen Programms stellt sich oft das Problem der Variablenkontrolle, d.h. welche Variablennamen dürfen noch verwendet werden bzw. wo könnten unerwünschte Überschneidungen zwischen mehreren Programmteilen auftreten?
Gelegentlich - bei sehr großen Programmen - wird man sogar die Anzahl der Variablen verringern müssen, d.h. man wird denselben Namen mehrfach im Programm für verschiedene Variablen verwenden müssen, da auch nur einmal angesprochene Variablen Speicherplatz benötigen. Der Überblick über zulässige Möglichkeiten der Änderung geht leicht verloren. Zudem ist häufig eine Dokumentation innerhalb des Programms durch REM-Statements aus Platzmangel nicht mehr möglich. Das Gleiche gilt für Änderungen von Zeilennummern, z.B. durch Zusammenfassen von Zeilen;
bei Zeilennummern, die als Sprungadressen im Programm auftauchen, ist dies nicht möglich ohne Änderung der entsprechenden Sprungbefehle.

Hier hilft nun ein weiteres Programmpaket weiter, mit dem man sich den Überblick über die Variablen seines Programms und die Sprungadressen verschaffen kann, mit dem man Umbenennungen vornehmen kann und mit dem man auch eine externe Dokumentation eines Programms in kleinem Umfang anlegen kann.

Dieses Paket heißt APPLE-DOC; das Copyright liegt bei Roger Wagner und der Firma Southwestern Data Systems in Kalifornien, USA. Es besteht aus 3 Teilen: VARDOC, LINEDOC und REPLACE. Drei Textfiles mit den genannten Namen können jeweils mit EXEC gestartet werden. Diese EXEC-files leisten nun zunächst einmal die Sicherung des zu bearbeitenden Programms. Die Speicherstellen 1Ø3 und 1Ø4 enthalten die Adresse des Programmbeginns und die Stellen 175 und 176 diejenige des Programmendes. Wird nun der Inhalt von 1Ø3 und 1Ø4 erhöht auf einen Wert größer als die Adresse des Programmendes, so ist das ursprüngliche Programm gesichert. Nun kann das eigentlich ausführende Programm geladen werden. Es sind dies die drei APPLESOFT-Programme VRF, LRF und RPL, von denen jeweils eins durch einen der drei genannten EXEC-files geladen und gestartet wird. Nach Abschluß der Analysen bzw. Änderungen können die alten Werte der genannten 4 Speicherstellen wiederhergestellt werden, da sie in den Stellen 1912 bis 1915 zwischengespeichert worden sind durch die Anweisungen in den jeweiligen EXEC-files.

7.3.1 Variablenkontrolle

Nach EXEC VARDOC ist die Neubelegung der oben genannten Speicherstellen erfolgt, die Sicherung ihrer Ursprungswerte und der Start des APPLESOFT-Programms VRF, das zunächst den Programmtitel mit Copyright auf dem Bildschirm anzeigt, danach einige Erläuterungen gibt und nach dem zu untersuchenden Programmbereich fragt.
Als Voreinstellung werden Anfang und Ende des Programms angenommen. Es folgt die Frage nach dem Namen des untersuchten Programms. Zu diesem Zeitpunkt ist die gewünschte Tabelle erstellt und durch Herabsetzen von HIMEM gegen Überschreiben gesichert worden.
Nach der Frage, ob der Drucker eingeschaltet werden soll, folgt die Präsentation des Ergebnisses: Eine Liste aller im Programm vorkommenden Variablennamen mit den Zeilennummern, in denen sie auftreten.
Nachfolgendes Beispiel stellt die Variablentabelle dar des in Abschnitt 2 schon eingeführten Programms zur Durchschnittsberechnung in seiner erweiterten Form.

```
*****************************
*                           *
*        DEMO VARDOC        *
*                           *
*  -->TABLE OF VARIABLES<-- *
*                           *
*****************************
```

```
A
100 100

A$
20 100 100 100 110 110 110 120

H$
100 100 110 110

H1
20 110 120

I
50 50 50

N
30 50 60 60

Q
120

S
50 50 60

X
40 50

END OF VAR. LIST
```

Danach kehrt das Programm zu seinem Menü zurück

```
DO YOU WANT TO: (1) RETURN TO PROGRAM
                (2) REVIEW LIST AGAIN
                (3) NEW ANALYSIS
                (4) ENTER DESCRIPTORS
                (5) SAVE TO DISK
                (6) GET FROM DISK
```

Die Option 1 bewirkt die Rückkehr zum bearbeiteten Programm, also die Beendigung von VARDOC mit Rücksetzung der Speicherstellen 1Ø3/1Ø4 und 175/176 auf die ursprünglichen Werte und die Heraufsetzung von HIMEM , d.h. Veränderung der Speicherstellen 115/116.

Option 2 gibt es in 3 Versionen: Die Eingabe von 2 führt zu derselben Bildschirmanzeige wie oben, erneut mit der Möglichkeit einer Druckerausgabe.

Die Eingabe von '2*' führt zu einer verkürzten Liste, die nur die Variablennamen - ohne zugehörige Zeilennummern - enthält:

```
*****************************
*                           *
*        DEMO VARDOC        *
*                           *
*  -->TABLE OF VARIABLES<-- *
*                           *
*****************************
```

```
A  A$  H$  H1  I  N  Q  S  X

END OF VAR. LIST
```

Die Eingabe von '2**' soll hier noch nicht demonstriert werden, aber auch dies ist möglich.

Option 3 löscht die alte Variablenliste, es ist möglich, ein weiteres Programm zu analysieren.

Die Optionen 4-6 sind besonders geeignet für die Dokumentation eines Programms. Option 4 ermöglicht die Eingabe von kurzen Erläuterungen zu den Variablennamen. In diesem Modus bedeutet RETURN statt einer Eingabe, daß die alte Beschreibung erhalten bleibt, wenn eine vorhanden war. Durch 'L' kann die Liste der Zeilennummern dieser Variablen abgerufen werden. Durch '/' wird eine alte Bezeichnung ersatzlos gelöscht. Neue Eingaben ersetzen alte Bezeichnungen. Durch 'X' kann die Eingabe vor Erreichen des Endes der Variablenliste abgebrochen werden.

Durch Option 5 können die Variablenbezeichnungen auf die Diskette geschrieben werden in ein Textfile, dessen Name aus dem eingegebenen Programmnamen und dem Zusatz VFILE besteht.
Durch Option 6 kann ein entsprechendes Textfile eingelesen werden.

Nach Aufruf der Option 4 kehrt das Programm ebenfalls zu dem bekannten Menü zurück. Die 3 Versionen der Option 2 bewirken jetzt - nach Eingabe von Variablenbezeichnungen - folgende Bildschirmanzeigen:

```
            ******************************
"2"   :     *                            *
            *        DEMO VARDOC         *
            *                            *
            *  -->TABLE OF VARIABLES<--  *
            *                            *
            ******************************

A - LAENGE DES TEXTES
100 100

A$ - AUSGABETEXT
20 100 100 100 110 110 110 120

H$ - HILFSSTRING AUS 40 BLANKS
100 100 110 110

H1 - NUMMER DER AUSGABEZEILE
20 110 120

I - ZAEHLER FUER EINGABE D. WERTE
50 50 50

N - ANZAHL DER WERTE
30 50 60 60

Q - SCHLEIFENINDEX
120

S - SUMME DER WERTE
50 50 60

X - WERT
40 50

END OF VAR. LIST
```

"2*" : Dies hat denselben Effekt wie vorher.

"2**" :

```
*****************************
*                           *
*        DEMO VARDOC        *
*                           *
*  -->TABLE OF VARIABLES<-- *
*                           *
*****************************
```

A - LAENGE DES TEXTES

A$ - AUSGABETEXT

H$ - HILFSSTRING AUS 40 BLANKS .

H1 - NUMMER DER AUSGABEZEILE

I - ZAEHLER FUER EINGABE D. WERTE

N - ANZAHL DER WERTE

Q - SCHLEIFENINDEX

S - SUMME DER WERTE

X - WERT

END OF VAR. LIST

7.3.2 Kontrolle der Sprungadressen

Nach EXEC LINEDOC geschieht das Gleiche wie bei VARDOC: Sicherung des Programms, Laden und Starten des APPLESOFT-Programms LRF, das einen Titel auf dem Bildschirm anzeigt, nach dem zu untersuchenden Programmabschnitt fragt, nach dem Namen des Programms und danach, ob der Drucker eingeschaltet werden soll.
Es wird eine Tabelle aller Zeilennummern angelegt, die als Sprungadressen im Programm auftauchen, und der jeweils zugehörigen Zeilen, aus denen die Sprungbefehle erfolgen.
Für das schon bekannte Beispiel der Durchschnittsberechnung sieht die Tabelle so aus:

```
*****************************
*                           *
*        DEMO LINEDOC       *
*                           *
*  -->TABLE OF LN# XREFS<-- *
*                           *
*****************************
```

```
    40
 50

   100
 20

   110
 130

 END OF LN# LIST
```

Auf dem Bildschirm sind die Sprungadressen (6Ø, 1ØØ, 11Ø) nicht nur eingerückt, sondern auch invers geschrieben, zur Unterscheidung von den aufrufenden Zeilen. Das Menü von LINEDOC sieht genauso aus wie das von VARDOC, Die Optionen bewirken genau das entsprechende:

```
DO YOU WANT TO: (1) RETURN TO PROGRAM
                (2) REVIEW LIST AGAIN
                (3) NEW ANALYSIS
                (4) ENTER DESCRIPTORS
                (5) SAVE TO DISK
                (6) GET FROM DISK
```

Auch hier gibt es Option 2 in den schon besprochenen 3 Versionen, die nach Abruf von Option 4 folgende Bildschirmanzeigen bewirken:

"2" :

```
*****************************
*                           *
*        DEMO LINEDOC       *
*                           *
*  -->TABLE OF LN# XREFS<-- *
*                           *
*****************************
```

```
   40 - EINGABE VON WERTEN
50

  100 - UNTERPROGRAMM LAUFZEILE
20

  110 - SCHLEIFENBEGINN FUER AUSDRUCK
130

END OF LN# LIST
```

"2*" :

```
*****************************
*                           *
*        DEMO LINEDOC       *
*                           *
*  -->TABLE OF LN# XREFS<-- *
*                           *
*****************************
```

```
40    100    110
END OF LN# LIST
```

"2**" :

```
*****************************
*                           *
*        DEMO LINEDOC       *
*                           *
*  -->TABLE OF LN# XREFS<-- *
*                           *
*****************************
```

```
 40 - EINGABE VON WERTEN

100 - UNTERPROGRAMM LAUFZEILE

110 - SCHLEIFENBEGINN FUER AUSDRUCK

END OF LN# LIST
```

7.3.3 Änderungen

Als Folge der Analysen mit Hilfe von VARDOC und LINEDOC könnte sich die Notwendigkeit von Änderungen innerhalb eines Programms herausgestellt haben. Zu diesem Zweck wird nun das dritte Programm im Paket APPLE-DOC eingesetzt. Durch EXEC REPLACE wird es initialisiert. Bevor das Programm RPL gestartet wird, erfolgt zunächst wie vorher die Sicherung des Ursprungsprogramms durch die Neubelegung der Speicherstellen 1Ø3/1Ø4. Dabei wird aber zusätzlich ein Puffer berücksichtigt, dessen Größe der Benutzer angeben kann in Einheiten von je 256 bytes. Dies ist notwendig, da durch die Änderungen das Programm möglicherweise länger wird, wenn z.B. die Variable A im gesamten Programm den neuen Namen A1 erhält. Ein weiterer Unterschied im Ablauf zu den ersten beiden Programmen ist der, daß nach dem Programmtitel als erstes das Menü erscheint; es enthält 6 Optionen:

```
DO YOU WANT TO:

  (1) REPLACE A VARIABLE/#
  (2) REPLACE A LITERAL
  (3) TRUNCATE & REPLACE A LITERAL
  (4) DOCUMENT A LITERAL
  (5) LIST A LINE
  (6) RETURN TO PROGRAM
```

Option 6 leistet ähnliches wie die Option 1 in den beiden vorigen Menüs, nämlich die "Freigabe" des Ursprungsprogramms. Mit Option 5 können Teile des Programms gelistet werden, wenn die vorzunehmenden Änderungen noch nicht klar sind.
Mit Option 1 können Variablennamen oder Sprungadressen ausgetauscht werden. Nach Eingabe von '1' erfolgt zunächst die Frage nach dem interessierenden Programmbereich.
Die beiden nächsten Eingaben beziehen sich auf Variablennamen oder Sprungadressen, und zwar auf deren "alte" und "neue" Werte, danach wird die Frage gestellt, ob der alte Wert im gesamten Programm oder nur in einzelnen Zeilen ersetzt werden soll:

```
WHAT CHAR'S TO REPLACE?A$
WHAT CHAR'S TO PUT THERE?B$
ALL OR SOME? (A/S):A
```

Durch diese Eingaben wird das Programm zur Durchschnittsberechnung wie folgt verändert:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HTAB 7: PRINT "BERECHNUNG DES DURCHSCHNITTS"
20 B$ = "DRUECKEN SIE IRGENDEINE TASTE":H1 = 20: GOSUB 100
30  INPUT "ANZAHL DER WERTE";N
40  INPUT "WERT";X
50 S = S + X:I = I + 1: IF I < N THEN 40
60  HOME : VTAB 10: PRINT "DURCHSCHNITT DER ";N;" WERTE: ";S
/ N
70  END
100 H$ = "                                        ":A =  LEN
(B$):B$ = B$ +  RIGHT$ (H$,40 - A)
110 H$ =  LEFT$ (B$,1):B$ =  RIGHT$ (B$,39) + H$: INVERSE :
VTAB (H1): CALL  - 958
120  VTAB (H1): PRINT  LEFT$ (B$,40): FOR Q = 1 TO 150: NEXT

130  IF  PEEK ( - 16384) < 128 THEN 110
140  NORMAL
150  HOME : RETURN
```

Im gesamten Programm wird die Variable A$ durch B$ ersetzt.

Wird auf die dritte Frage "ALL OR SOME?" mit 'S' geantwortet, so wird der Benutzer zu jeder Ersetzung zunächst befragt und es wird nur ersetzt, wenn er es wünscht; dieses Vorgehen wird sinnvoll sein bei irrtümlich aufgetretenen Doppelbenennungen, wie sie etwa nach dem Mischen zweier verschiedener Programme denkbar sind.
Es muß noch darauf hingewiesen werden, daß Arraynamen durch eine dem eigentlichen Namen nachfolgende linke Klammer kenntlich gemacht werden müssen.
Statt eines Variablennamens kann durch die Option 1 auch eine Sprungadresse durch eine andere ersetzt werden, wie dies etwa nach der Zusammenfassung von Zeilen oder nach dem Löschen von REM-Statements nötig werden kann.

Option 2 ermöglicht es, Strings im Programm suchen und ersetzen zu lassen. Nach dem Erfragen des interessierenden Programmbereichs muß zunächst die Frage "ALL OR SOME?" beantwortet werden. Dann muß hinter der Zeilennummer 74Ø der "alte" zu ersetzende String eingegeben werden, hinter der Zeilennummer 75Ø der "neue". Der Programmablauf bleibt unterbrochen bis das Kommando

GOTO 76Ø

über die Tastatur eingetippt wird. Als Beispiel wurde in 74Ø DURCHSCHNITT eingegeben und in 75Ø MITTELWERT; mit der Option ALL erhält man folgendes Listing (die Zeilen 3Ø und 8Ø sind geändert worden):

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HTAB 7: PRINT "BERECHNUNG DES MITTELWERTS"
20 B$ = "DRUECKEN SIE IRGENDEINE TASTE":H1 = 20: GOSUB 100
30  INPUT "ANZAHL DER WERTE";N
40  INPUT "WERT";X
50 S = S + X:I = I + 1: IF I < N THEN 40
60  HOME : VTAB 10: PRINT "MITTELWERT DER ";N;" WERTE: ";S /
N
70  END
100 H$ = "                                        ":A =  LEN
(B$):B$ = B$ +  RIGHT$ (H$,40 - A)
110 H$ =  LEFT$ (B$,1):B$ =  RIGHT$ (B$,39) + H$: INVERSE :
VTAB (H1): CALL  - 958
120  VTAB (H1): PRINT  LEFT$ (B$,40): FOR Q = 1 TO 150: NEXT

130  IF  PEEK ( - 16384) < 128 THEN 110
140  NORMAL
150  HOME : RETURN
```

Option 3 ist nützlich zum Entfernen von REM-Zeilen, wenn bei sehr großen Programmen der Speicherplatz knapp wird; es wird nicht nur der in Zeile 74Ø angegebene String ersetzt, sondern auch alles was rechts von ihm in derselben Zeile steht.
Die Option ist automatisch 'SOME'

Beispiel:

```
74Ø  REM
75Ø  :
```

Es wird auf Wunsch nicht nur der Befehl 'REM' durch ':" ersetzt, sondern auch der rechts von REM stehende Kommentar "abgeschnitten" (truncated).

Die Option 4 schließlich macht es möglich, vor etwaigen Änderungen zunächst eine Liste aller Zeilen anfertigen zu lassen, in denen ein bestimmter String auftaucht. Dabei kann auch der Drucker nach Bedarf wieder eingeschaltet werden.

Damit sind die Möglichkeiten des Programmpakets APPLE-DOC im wesentlichen wohl erläutert worden. Nur bei sehr umfangreichen Programmen könnte der Einsatz dieser Hilfsroutinen problematisch werden, da natürlich für die Routinen und die erstellten Tabellen zusätzlich Platz benötigt wird. Die Programme benötigen ca. 6 Kbyte; der Platzbedarf der Tabellen kann nicht allgemein angegeben werden, da diese Größe natürlich vom jeweils untersuchten Programm abhängt. Sollte es Platzprobleme geben, können diese auf 2 Arten gelöst werden:

1. Kürzen der Hilfsprogramme auf die unbedingt notwendige Größe (etwa 3.3K bei VRF).

2. Löschen von Teilen des Ursprungsprogramms aus dem Arbeitsspeicher, die dann in einem oder mehreren weiteren Läufen analysiert werden könnten.

7.4 Initialisierung

In Kapitel 2 wurde schon kurz erläutert, daß jede fabrikneue Diskette zunächst initialisiert werden muß; dies erfolgt durch den Befehl

INIT name

Bei Durchführung dieses Befehls geschieht dreierlei:

1. Die zunächst völlig leere Diskette wird "formatiert", d.h. sie wird eingeteilt in Spuren (das sind konzentrische Ringe um das mittlere Loch; sie werden auch tracks genannt) und Sektoren (das sind Abschnitte in den einzelnen Spuren).
   Die Formatierung des APPLE II erzeugt 35 Spuren zu je 16 Sektoren, jeder Sektor umfaßt 256 bytes. Die Spuren sind hexadezimal von außen nach innen durchnumeriert durch $Ø bis $22, die Sektoren jeweils von $Ø bis $F.

2. DOS wird auf die Diskette geschrieben, und zwar in die tracks $Ø bis $2. Die Spur $11 wird für das Inhaltsverzeichnis der Diskette eingerichtet.

3. Das Begrüßungsprogramm wird unter dem hinter INIT angegebenen Namen abgespeichert.

Nach der Initialisierung stehen dem Benutzer somit 31 Spuren zu 16 Sektoren zu je 256 bytes zur Verfügung, d.h. Speicherplatz von insgesamt 124K, abzüglich des meist recht geringen Platzes, den das Begrüßungsprogramm benötigt. Dieses kann sogar nur aus einer BASIC-Zeile mit einem einzigen Befehl bestehen. Es ist jedoch durchaus sinnvoll, das Begrüßungsprogramm etwas umfangreicher zu programmieren.
Zwar hat es zunächst im wesentlichen nur die Aufgabe, die Diskette "booten" zu können, d.h. auch mit einer anderen als der APPLE-Masterdiskette DOS laden zu können, doch können ihm auch andere Aufgaben zugewiesen werden. Dieser Abschnitt soll dazu einige Anregungen liefern. Der Leser kann dann seine ganz persönlichen Wünsche für die Disketteninitialisierung sicher leicht verwirklichen. Im folgenden soll das Begrüßungsprogramm immer den Namen HELLO erhalten. Dieser Name ist weit verbreitet, er findet sich z.B. auch auf der Systemdiskette.

Eine erste Aufgabe von HELLO kann darin bestehen, Informationen über die eingelegte Diskette zu liefern, d.h. wann wurde sie initialisiert, welche Nummer oder welchen Namen trägt sie und welchen Inhalt hat sie. Man kann etwa so vorgehen, daß man folgendes Programm mit den Namen INITIAL auf einer Diskette abspeichert:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  PRINT "DIESES PROGRAMM DIENT DER INITIALISIERUNG. DAZU MUSS
DIE ZEILE 5 GELOESCHT WERDEN UND DIE GITTER IN ZEILE 20 MUESSEN
ERSETZT WERDEN.": END
10  TEXT : FOR I = 1 TO 41: PRINT "*";: NEXT : GOSUB 999: HTAB
8: PRINT "R. PRUST / W. VOSS";: GOSUB 999
20  HTAB 8: PRINT "DISKETTE NR. ##";: GOSUB 999: HTAB 8: PRINT
"INITIALISIERT AM ##.##.####";
30  GOSUB 999: HTAB 40: FOR I = 1 TO 41: PRINT "*";: NEXT :
PRINT : FOR I = 1 TO 2500: NEXT : HOME
40  VTAB 2: PRINT "CATALOG": END
999  HTAB 40: PRINT "**";: HTAB 40: PRINT "**";: RETURN
```

Wenn man dieses Programm startet, erscheint folgender Bildschirmausdruck:

```
APPLE-TIPS UND TRICKS

R.PRUST / W.VOSS 1984

DIESES PROGRAMM DIENT DER INITIALISIERUN
G. DAZU MUSS DIE ZEILE 5 GELOESCHT WERDE
N UND DIE GITTER IN ZEILE 20 MUESSEN ERS
ETZT WERDEN.
```

Nachdem man die Zeile 5 gelöscht hat, müßte man - am einfachsten mit Hilfe des PROGRAM LINE EDITOR - die Zeile 2Ø ändern. Ansonsten sieht das Programm so aus:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  TEXT : FOR I = 1 TO 41: PRINT "*";: NEXT : GOSUB 999: HTAB
8: PRINT "R. PRUST / W. VOSS";: GOSUB 999
20  HTAB 8: PRINT "DISKETTE NR. ##";: GOSUB 999: HTAB 8: PRINT
"INITIALISIERT AM ##.##.####";
30  GOSUB 999: HTAB 40: FOR I = 1 TO 41: PRINT "*";: NEXT :
PRINT : FOR I = 1 TO 2500: NEXT : HOME
40  VTAB 2: PRINT "CATALOG": END
999  HTAB 40: PRINT "**";: HTAB 40: PRINT "**";: RETURN
```

Die Zeilen 1Ø bis 3Ø bewirken zunächst eine Bildschirmanzeige:

```
****************************************
*                                      *
*                                      *
*      R. PRUST / W. VOSS              *
*                                      *
*      DISKETTE NR. ##                 *
*                                      *
*      INITIALISIERT AM ##.##.####     *
*                                      *
*                                      *
****************************************
```

Dann folgt eine leere FOR...NEXT-Schleife der Länge 25ØØ, die dafür sorgt, daß dieser Ausdruck eine gewisse Zeit auf dem Schirm stehen bleibt. Ohne Eingriff des Benutzers geht es dann im Ablauf weiter, zunächst mit dem HOME-Befehl am Ende von Zeile 3Ø. Durch Zeile 4Ø wird nun programmgesteuert der Catalog der Diskette abgerufen. Es muß darauf hingewiesen werden, daß in dem String vor dem "C" von CATALOG ein CTRL-D eingegeben wurde, das aber im Druckerlisting natürlich nicht sichtbar ist.
Der Befehl hätte auch die Form

```
PRINT CHR$(4);"CATALOG"
```

haben können. Die Zeile 999 ist ein Unterprogramm, das für die Sternchen am linken und rechten Rand des Bildschirms verantwortlich ist.

Jeder Benutzer kann natürlich eine andere Bildschirmanzeige für wünschenswert halten. Es scheint jedoch sinnvoll, da es Zeit einspart, ein Initialisierungsprogramm zur Verfügung zu haben, in dem nur einige wenige Änderungen zu machen sind für die aktuell zu initialisierende Diskette. Falls das entsprechende Programm INITIAL sehr viele aktuelle Informationen benötigt - nicht nur das Datum und eine Zahl - wäre es vielleicht zweckmäßig, ein Programm zu erstellen, das alle erforderlichen Eingaben abfragt und dann seinerseits ein Programm schreibt, nämlich ein Textfile INITIAL, das mit EXEC in den Speicher geladen werden könnte. Das Vorgehen sei am folgenden Beispiel demonstriert:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM    PROGRAMM ZUR ERSTELLUNG EINES
20  REM        INITIALISIERUNGSPROGRAMMS
30  INPUT "DATUM (TT.MM.JJJJ) ";A$: IF  MID$ (A$,3,1) <  > "."
OR  MID$ (A$,6,1) <  > "." THEN 30
40  INPUT "NUMMER DER DISKETTE ";N$
50 D$ =  CHR$ (4)
60  PRINT D$;"OPEN INITIAL"
70  PRINT D$;"WRITE INITIAL"
80  PRINT "10 TEXT:FORI=1TO41:?"; CHR$ (34);"*"; CHR$ (34);";:
NEXT : GOSUB999 :HTAB 8 :? "; CHR$ (34);"RENATE PRUST"; CHR$
(34);"; : GOSUB 999"
90  PRINT "20 HTAB 8 : ?"; CHR$ (34);"DISKETTE NR. ";N$; CHR$
(34);"; : GOSUB 999 : HTAB 8 : ? "; CHR$ (34);"INITIALISIERT
AM ";A$; CHR$ (34);";"
100  PRINT "30 GOSUB 999 : HTAB 40 : FOR I=1 TO 41 : ? "; CHR$
(34);"*"; CHR$ (34);"; : NEXT : ? : FOR I=1 TO 2500: NEXT :
HOME"
110  PRINT "40 VTAB2:?"; CHR$ (34);"CATALOG"; CHR$ (34);":END"
120  PRINT "999 HTAB 40 : ? "; CHR$ (34);"**"; CHR$ (34);";
: HTAB 40 : ? "; CHR$ (34);"**"; CHR$ (34);"; : RETURN"
130  PRINT D$;"CLOSE INITIAL"
140  END
```

Das vorstehende Programm MAKE INITIAL erzeugt ein Textfile mit dem Namen INITIAL, so daß nach dem Kommando EXEC INITIAL die schon bekannten Zeilen 1Ø-999 im Arbeitsspeicher zur Verfügung stehen - mit dem Unterschied, daß anstelle der "#"-Zeichen ein Datum und eine Diskettennummer nach Wunsch des Benutzers eingesetzt wurden

Was leistet das Programm MAKE INITIAL im einzelnen?

Programmbeschreibung

Sätze	Inhalt
1Ø- 2Ø	Erläuterung
3Ø	Eingabe des aktuellen Datums im angegebenen Format als A$; kleine Fehlerabfangroutine
4Ø	Eingabe der Diskettennummer als N$
5Ø	Zuordnung von CTRL-D zu D$
6Ø	Öffnen der Datei INITIAL
7Ø	Anweisung, in die Datei zu schreiben - alle nachfolgenden PRINT-Befehle werden an die Diskettenstation weitergeleitet
8Ø-12Ø	Die Zeilen 1Ø-999 des erkannten Programms INITIAL werden als Text eines PRINT-Befehls auf die Diskette geschrieben. Dabei müssen die Anführungszeichen, etwa um den Namen RENATE PRUST, durch ihre ASCII-Kennziffer dargestellt werden, d.h. durch CHR$(34). Weiter ist zu beachten, daß bei Ausführung des Programms MAKE INITIAL die Variablen N$ und A$ durch ihre zuvor eingegebenen Werte (Zeilen 3Ø und 4Ø) ersetzt werden.
13Ø	Die Datei wird geschlossen
14Ø	Ende des Programms

Das gewünschte Begrüßungsprogramm mit den aktuellen Informationen über die Diskette steht nach folgenden Kommandos zur Verfügung, wenn MAKE INITIAL sich auf einer Diskette befindet:

```
RUN MAKE INITIAL
-
NEW
EXEC INITIAL
```

Durch LIST kann dies kontrolliert werden.
Der Befehl NEW ist notwendig, da durch EXEC im Gegensatz zu RUN das alte Programm nicht gelöscht wird.
Nur Zeilen des alten Programms, deren Nummern mit Zeilennummern des neuen Programms übereinstimmen, werden überschrieben, alle anderen bleiben erhalten.
Ohne NEW entstünde ein "Programmsalat" aus MAKE INITIAL und INITIAL, da das zweite Programm kürzer ist als das erste.

Zur eigenen Bequemlichkeit kann man nun das Programm MAKE INITIAL noch so erweitern, daß der Befehl

```
RUN MAKE INITIAL
```

allein genügt, um das gewünschte Begrüßungsprogramm zwecks Initialisierung einer neuen Diskette zur Verfügung zu haben.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM   PROGRAMM ZUR ERSTELLUNG EINES
20  REM       INITIALISIERUNGSPROGRAMMS
30  INPUT "DATUM (TT.MM.JJJJ) ";A$: IF  MID$ (A$,3,1) <  > "."
OR  MID$ (A$,6,1) <  > "." THEN 30
40  INPUT "NUMMER DER DISKETTE ";N$
50 D$ =  CHR$ (4)
60  PRINT D$;"OPEN INITIAL"
70  PRINT D$;"WRITE INITIAL"
80  PRINT "10 TEXT:FORI=1TO41:?"; CHR$ (34);"¥"; CHR$ (34);";:
NEXT : GOSUB999 :HTAB 8 :? "; CHR$ (34);"RENATE PRUST"; CHR$
(34);"; : GOSUB 999"
90  PRINT "20 HTAB 8 : ?"; CHR$ (34);"DISKETTE NR. ";N$; CHR$
(34);"; : GOSUB 999 : HTAB 8 : ? "; CHR$ (34);"INITIALISIERT
AM ";A$; CHR$ (34);";"
100  PRINT "30 GOSUB 999 : HTAB 40 : FOR I=1 TO 41 : ? "; CHR$
(34);"¥"; CHR$ (34);"; : NEXT : ? : FOR I=1 TO 2500: NEXT :
HOME"
110  PRINT "40 VTAB2:?"; CHR$ (34);"CATALOG"; CHR$ (34);":END"
120  PRINT "999 HTAB 40 : ? "; CHR$ (34);"¥¥"; CHR$ (34);";
: HTAB 40 : ? "; CHR$ (34);"¥¥"; CHR$ (34);"; : RETURN"
130  PRINT D$;"CLOSE INITIAL"
140  PRINT D$;"EXEC INITIAL"
150  DEL 50,150
```

Die Zeilen 1Ø bis 12Ø sind identisch mit der obigen Version des Programms. Durch Zeile 13Ø werden aber nun programmgesteuert die soeben geschriebenen Zeilen des Textfiles INITIAL (1Ø-4Ø,999) geladen. Anschließend müssen die nun unnötigen Zeilen des alten "MAKE INITIAL" gelöscht werden. Dies geschieht durch die Zeile 14Ø.

Nach einem DEL-Befehl im Programm wird die Programmausführung beendet, so daß auf ein zusätzliches END verzichtet werden kann.
Nach RUN MAKE INITIAL kann man sich durch LIST davon überzeugen, daß nur die gewünschten Zeilen im Arbeitsspeicher stehen:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  TEXT : FOR I = 1 TO 41: PRINT "*";: NEXT : GOSUB 999: HTAB
8: PRINT "R. PRUST / W. VOSS";: GOSUB 999
20  HTAB 8: PRINT "DISKETTE NR. 17";: GOSUB 999: HTAB 8: PRINT
"INITIALISIERT AM 11.05.1984";
30  GOSUB 999: HTAB 40: FOR I = 1 TO 41: PRINT "*";: NEXT :
PRINT : FOR I = 1 TO 2500: NEXT : HOME
40  VTAB 2: PRINT "CATALOG": END
999  HTAB 40: PRINT "**";: HTAB 40: PRINT "**";: RETURN
```

Über solche Bildschirmanzeigen hinaus kann man aber noch anderes in sein Begrüßungsprogramm aufnehmen. Sehr sinnvoll erscheint uns beispielsweise die Aufnahme des Editors aus Kapitel 6 oder anderer Hilfsroutinen. Dazu geht man folgendermaßen vor:

In den leeren Arbeitsspeicher wird der Editor geladen durch

```
LOAD PROGRAM LINE EDITOR
```

Die überflüssigen Zeilen werden gelöscht durch

```
DEL 1Ø,3Ø
```

Nun kann man das Programm INITIAL dazuladen, entweder durch EXEC INITIAL, wenn der Weg über das Textfile gewählt wurde, oder mit Hilfe von APA durch

```
&H
LOAD INITIAL
&M
```

In beiden Fällen sieht das Programm dann so aus:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  TEXT : FOR I = 1 TO 41: PRINT "*";: NEXT : GOSUB 999: HTAB
8: PRINT "R. PRUST / W. VOSS";: GOSUB 999
20  HTAB 8: PRINT "DISKETTE NR. 17";: GOSUB 999: HTAB 8: PRINT
"INITIALISIERT AM 11.05.1984";
30  GOSUB 999: HTAB 40: FOR I = 1 TO 41: PRINT "*";: NEXT :
PRINT : FOR I = 1 TO 2500: NEXT : HOME
40  VTAB 2: PRINT "CATALOG": END
999  HTAB 40: PRINT "**";: HTAB 40: PRINT "**";: RETURN
63999  CALL  PEEK (175) +  PEEK (176) * 256 - 1822: RETURN :
REM  "
```

```
* PROGRAM LINE EDITOR V2.0 *

      COPYRIGHT (C) 1980
         NEIL KONZEN
```

Nun wird das END in Zeile 4Ø gelöscht und eine neue Zeile 5Ø eingefügt:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  TEXT : FOR I = 1 TO 41: PRINT "*";: NEXT : GOSUB 999: HTAB
8: PRINT "R. PRUST / W. VOSS";: GOSUB 999
20  HTAB 8: PRINT "DISKETTE NR. 17";: GOSUB 999: HTAB 8: PRINT
"INITIALISIERT AM 11.05.1984";
30  GOSUB 999: HTAB 40: FOR I = 1 TO 41: PRINT "*";: NEXT :
PRINT : FOR I = 1 TO 2500: NEXT : HOME
40  VTAB 2: PRINT "CATALOG"
50  GOSUB 63999: PRINT "FP"
999  HTAB 40: PRINT "**";: HTAB 40: PRINT "**";: RETURN
63999  CALL  PEEK (175) +  PEEK (176) * 256 - 1822: RETURN :
REM  "
```

```
* PROGRAM LINE EDITOR V2.0 *

      COPYRIGHT (C) 1980
         NEIL KONZEN
```

Nach dem Booten einer Diskette mit diesem Begrüßungsprogramm HELLO steht der EDITOR ebenfalls zur Verfügung.

Der Befehl PRINT "FP" in Zeile 5Ø beendet das Programm, er ersetzt das NEW und führt zu APPLESOFT-BASIC zurück.

Wenn man alle seine Disketten mit diesem oder einem ähnlichen HELLO initialisiert hat, hat man sofort nach dem Einschalten die Möglichkeit, mit dem Editor zu arbeiten. Das ist bequemer als das Laden zwischendurch. Es soll aber noch einmal erwähnt werden, daß HIMEM, d.h. die größtmögliche von BASIC adressierbare Speicherstelle, dadurch auf einen kleineren Wert gesetzt wird. Nach dem Booten von DOS und dem Editor stehen nur noch 36K Arbeitsspeicher zur Verfügung. Will man noch weitere Hilfsroutinen in HELLO einbauen, so ist auch das möglich, allerdings nicht ganz so einfach wie vorher. Als Beispiel soll APA herangezogen werden.

Um dieses sehr nützliche Hilfsprogramm anwenden zu können, muß einfach der Befehl

```
RUN LOADAPA
```

gegeben werden. Außerdem werden aber von LOADAPA die Dateien RBOOT, RLOAD und APA angesprochen, die also ebenfalls auf derselben Diskette zur Verfügung stehen müssen.

Hat man zwei Laufwerke zur Verfügung, könnte man folgendermaßen vorgehen:
In das 2. Laufwerk kommt beim Booten durch PR#6 oder beim Einschalten des Rechners immer die Diskette mit dem DOS TOOL KIT, das auch die vier benötigten APA-files umfaßt. Im 1. Laufwerk liegt eine durch das nachstehende HELLO initialisierte Diskette:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  TEXT : FOR I = 1 TO 41: PRINT "*";: NEXT : GOSUB 999: HTAB
8: PRINT "R. PRUST / W. VOSS";: GOSUB 999
20  HTAB 8: PRINT "DISKETTE NR. 17";: GOSUB 999: HTAB 8: PRINT
"INITIALISIERT AM 11.05.1984";
30  GOSUB 999: HTAB 40: FOR I = 1 TO 41: PRINT "*";: NEXT :
PRINT : FOR I = 1 TO 2500: NEXT : HOME
40  VTAB 2: PRINT "CATALOG"
50  GOSUB 63999: PRINT "RUN LOADAPA,D2"
999  HTAB 40: PRINT "**";: HTAB 40: PRINT "**";: RETURN
63999  CALL  PEEK (175) +  PEEK (176) * 256 - 1822: RETURN :
REM  "
```

```
* PROGRAM LINE EDITOR V2.0 *

     COPYRIGHT (C) 1980
        NEIL KONZEN
```

Die PRINT-Befehle in den Zeilen 4Ø und 5Ø enthalten als erstes Zeichen ein CTRL-D, das im Druckerlisting nicht zu sehen ist. Beim Booten beginnen zunächst die bekannten Vorgänge: die Bildschirmanzeige über die Nummer der Diskette, der Catalog und das Starten des Editors.
Zusätzlich wird aber in dieser Version von HELLO das Programm LOADAPA auf der Diskette im Laufwerk 2 gestartet.
Danach endet das Begrüßungsprogramm, im Speicher befindet sich LOADAPA und der Laufwerkpointer steht auf 2.

Will man dies vermeiden, könnte man LOADAPA abändern, und zwar die Zeile 33Ø. Statt END sollte jetzt der CATALOG-Befehl für Drive 1 erfolgen und anschließend könnte durch

```
DEL 1Ø,36Ø
```

das Programm selbst gelöscht werden.
Falls ein Fehler während des Ladens von APA aufgetreten ist, wird die Zeile 34Ø angesprungen. Dieser letzte Teil des Programms sollte also unverändert bleiben. Es sollte ein Neustart möglich sein, deshalb wird das END in Zeile 36Ø nicht durch DEL ersetzt:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
100  REM  PROGRAM TO LOAD APA
110  HOME
120  ONERR  GOTO 340
130 ADRS = 0
140  PRINT  CHR$ (4),"BLOAD RBOOT"
150  CALL 520: REM  EXECUTE RBOOT
160 ADRS =  USR (0),"APA"
170  REM   BRING IN APA, ADRS=STARTING ADDRESS
180  CALL ADRS: REM   INITIALIZE APA
190  PRINT : PRINT
200  PRINT "&RENUMBER <START>,<INC>,<FIRST>,<LAST>"
210  PRINT "&HOLD"
220  PRINT "&MERGE"
230  PRINT "&LENGTH"
240  PRINT "&COMPRESS"
250  PRINT "&SHOW"
260  PRINT "&NOSHOW"
270  PRINT "&AUTO <START>,<INC>"
280  PRINT "&MANUAL"
290  PRINT "&XREF"
300  PRINT "&KEYS"
310  PRINT : PRINT
320  POKE 216,0: REM  ON ERR OFF
330  PRINT "CATALOG,D1"
335  DEL 100,360
340  PRINT "UNABLE TO LOAD"
350  POKE 216,0: REM  ON ERR OFF
360  END
```

Die Zeilen 19Ø bis 31Ø zeigen alle APA-Befehle. Ist man mit ihnen vertraut, kann man diese Zeilen löschen. Dieses gekürzte LOADAPA kann man dann in das Begrüßungsprogramm einmischen, das jetzt so aussieht:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  TEXT : FOR I = 1 TO 41: PRINT "*";: NEXT : GOSUB 999: HTAB
8: PRINT "R. PRUST / W.VOSS";: GOSUB 999
20  HTAB 8: PRINT "DISKETTE NR. 17";: GOSUB 999: HTAB 8: PRINT
"INITIALISIERT AM 11.05.1984";
30  GOSUB 999: HTAB 40: FOR I = 1 TO 41: PRINT "*";: NEXT :
PRINT : FOR I = 1 TO 2500: NEXT : HOME
50  GOSUB 63999
100  REM  PROGRAM TO LOAD APA
110  HOME
120  ONERR  GOTO 220
130 ADRS = 0
140  PRINT  CHR$ (4),"BLOAD RBOOT"
150  CALL 520: REM  EXECUTE RBOOT
160 ADRS =  USR (0),"APA"
170  REM   BRING IN APA, ADRS=STARTING ADDRESS
180  CALL ADRS: REM   INITIALIZE APA
190  POKE 216,0: REM  ON ERR OFF
200  PRINT "CATALOG,D1"
210  DEL 10,63999
220  PRINT "UNABLE TO LOAD"
230  POKE 216,0: REM  ON ERR OFF
240  END
999  HTAB 40: PRINT "**";: HTAB 40: PRINT "**";: RETURN
63999  CALL  PEEK (175) +  PEEK (176) * 256 - 1822: RETURN :
REM  "
```

```
* PROGRAM LINE EDITOR V2.0 *

     COPYRIGHT (C) 1980
        NEIL KONZEN
```

Wichtig ist, daß immer zuerst der Editor gestartet werden muß durch GOSUB 63999, danach kann dann APA geladen werden.

Hat man nur ein Laufwerk zur Verfügung, können die Laufwerksadressierungen D1 bzw. D2 in den Diskettenbefehlen entfallen.
Nach der Initialisierung einer Diskette durch eine der beiden letzten HELLO-Versionen müssen aber die Dateien APA, RBOOT und RLOAD auf die neue Diskette kopiert werden, da LOADAPA sie anspricht (genau gesagt, wird RLOAD von RBOOT aufgerufen). Dies kann mit dem File Developper von der Systemdiskette erfolgen durch

```
BRUN FID
```

Ob man nun APA oder eine andere Hilfsroutine sofort mit in das Begrüßungsprogramm einbaut - durch RUN oder durch Einmischen - wird davon abhängen, wie oft man solche Routinen benötigt. Der Arbeitsstil und die Aufgabenstellungen des jeweiligen Benutzers sind also letztlich entscheidend für den Umfang des Begrüßungsprogramms. Den Editor wird wohl jeder benötigen, da zumindest Tippfehler immer auftauchen.
Alles andere hängt sehr von den Wünschen des Einzelnen ab, so daß wir mit APA nur ein Demonstrationsbeispiel vorstellen wollten, wie Initialisierungsprogramme erweitert werden können.

Es ist ja auch zu bedenken, daß Speicherplatz verloren geht, z.B. durch sofortiges Laden von APA, und zwar im Arbeitsspeicher und auf der Diskette.

Wichtig ist aber auf jeden Fall, daß die Routinen miteinander verträglich sein müssen. Es muß möglicherweise - wie beim Editor und APA - auf die Reihenfolge beim Laden geachtet werden.

7.5 Bildschirmgestaltung

Die Aufgaben der Bildschirmgestaltung ist meist kein sehr schwieriges Problem, aber sehr zeitaufwendig. Aus diesem Grunde haben wir einige Anregungen zu diesem Bereich in das vorliegende Buch aufgenommen, durch die der Leser sich möglicherweise manche zeitraubende Vorüberlegungen ersparen kann.

7.5.1 Positionierung auf dem Bildschirm

Die wichtigsten Befehle wurden schon genannt und sollen hier im Zusammenhang noch einmal an Beispielen dargestellt werden.
Zunächst sind dies die BASIC-Befehle

HTAB und VTAB

und die TAB-Funktion als Ergänzung des PRINT-Befehls. In den bisher vorgestellten Beispielen wurden diese Positionierungs-Statements schon verwendet. Deshalb soll hier nur an einem kleinen Programm ihre Wirkungsweise aufgezeigt werden, insbesondere sollen dabei die Vorteile von HTAB und VTAB herausgestellt werden. Solche Cursorpositionierungs-Statements sind nicht auf allen Rechnern verfügbar; der APPLE bietet da schon einen gewissen Komfort.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HOME : PRINT  TAB( 10)"DEMONSTRATIONSPROGRAM"
20  PRINT  TAB( 6)"FUER DIE BILDSCHIRMGESTALTUNG"
30  PRINT : PRINT : PRINT : PRINT : PRINT
40  PRINT  TAB( 25)"EINGABE: ";: INPUT "";A$
50  PRINT : PRINT : PRINT
60  PRINT "DIES SOLL ZUNAECHST GENUEGEN.": PRINT : PRINT : PRINT

70  PRINT  TAB( 8)"TSCHUESS"; TAB( 25)"BIS BALD"
80  END
```

Für die horizontale Position auf dem Bildschirm wird TAB eingesetzt, die vertikale kann nur relativ zur aktuellen Position durch "leere" PRINT-Statements und nur nach unten bestimmt werden - ebenso wie TAB nur nach rechts Wirkung hat, wie etwa in Zeile 7Ø. Ist dort z.B. der Wert für die zweite TAB-Funktion in der Zeile kleiner als der erste, so geschieht kein Sprung nach links, sondern das TAB wird ignoriert.

Durch VTAB und HTAB können Zeilen und Spalten direkt angesteuert werden durch Angabe ihrer Nummer (1 bis 4Ø bzw. 1 bis 24) und nicht nur von oben nach unten bzw. von links nach rechts. So kann die Benutzung von VTAB einige leere PRINT-Befehle ersetzen.

```
3  HOME : HTAB 9: PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB 9: PRINT "R.PRUST / W.VOSS 1984": PRINT

10  HOME : HTAB 10: PRINT "DEMONSTRATIONSPROGRAM"
20  HTAB 6: PRINT "FUER DIE BILDSCHIRMGESTALTUNG"
40  VTAB 8: HTAB 25: INPUT "EINGABE: ";A$
60  VTAB 12: PRINT "DIES SOLL ZUNAECHST GENUEGEN."
70  VTAB 16: HTAB 8: PRINT "TSCHUESS"; TAB( 25)"BIS BALD"
80  END
```

An Zeile 7Ø erkennt man, daß HTAB und TAB auch zusammen für eine Zeile verwendet werden können.

Die beiden folgenden Beispielprogramme erklären sich im wesentlichen selbst. Damit die Reihenfolge der Bildschirmtexte deutlicher wird, wurde in Zeile 6 der Befehl SPEED=1ØØ aufgenommen.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 2000: NEXT
6  SPEED= 100
10  HOME : HTAB 10: PRINT "DEMONSTRATIONSPROGRAM"
20  HTAB 6: PRINT "FUER DIE BILDSCHIRMGESTALTUNG"
40  VTAB 8: HTAB 25: INPUT "EINGABE: ";A$
60  VTAB 12: PRINT "DIES SOLL ZUNAECHST GENUEGEN."
70  VTAB 22: PRINT "WIR KOENNEN NUN AUCH HIER ETWAS SCHREI-":
PRINT "BEN UND NACHHER WIEDER NACH OBEN GEHEN !";
80  VTAB 12: CALL  - 868: HTAB 13: PRINT "HIER WIRD JETZT GELOESCHT"
90  SPEED= 255: END
```

CALL-868 in Zeile 8Ø löscht die aktuelle Bildschirmzeile von der Cursorposition ab nach rechts.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
5  FOR I = 1 TO 2000: NEXT
6  SPEED= 100
10  HOME : HTAB 10: PRINT "DEMONSTRATIONSPROGRAM"
20  HTAB 6: PRINT "FUER DIE BILDSCHIRMGESTALTUNG"
30  VTAB 15: PRINT "MIT VTAB HABEN SIE DIE MOEGLICHKEIT,": PRINT
"ERST IM UNTEREN TEIL DES SCHIRMS EINE": PRINT "MITTEILUNG ZU
MACHEN UND DANN WIEDER": PRINT "FUER DAS INPUT NACH OBEN ZU
GEHEN."
35  PRINT "AEHNLICHES GILT FUER HTAB, WIE SIE": PRINT "GLEICH
SEHEN WERDEN."
40  VTAB 8: HTAB 25: PRINT "EINGABE: ";
50  HTAB 9: PRINT "MACHEN SIE EINE ";
60  HTAB 34: INPUT "";A$
70  VTAB 22: SPEED= 255: END
```

7.5.2 Stringfunktionen

Wenn über Bildschirmgestaltung geredet wird, müssen in der Regel auch die Stringfunktionen mit einbezogen werden. Denn der Bildschirminhalt soll ja meist nicht nur aus Zahlen bestehen, sondern auch erläuternde Texte zu den Zahlen umfassen.
Auch in früheren Beispielen wurden diese Funktionen gelegentlich schon eingesetzt.

Das folgende Beispiel enthält eine kleine Spielerei, die Rotation von Zeichen innerhalb eines Strings; das erste Zeichen wird jeweils weggenommen und hinten wieder angehängt, solange bis der ursprüngliche String wieder zusammengesetzt ist.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HOME : INPUT "ZEICHENKETTE ";X$
20 X$ = X$ + " ":X =  LEN (X$)
30  HOME : PRINT X$
40  FOR I = 1 TO X - 1
50 Y$ =  LEFT$ (X$,1)
60 X$ =  RIGHT$ (X$,X - 1) + Y$
70  PRINT X$
80  NEXT
90  END
```

Der Bildschirmausdruck für die Zeichenkette "Geburtstag" sieht so aus:

```
GEBURTSTAG
EBURTSTAG G
BURTSTAG GE
URTSTAG GEB
RTSTAG GEBU
TSTAG GEBUR
STAG GEBURT
TAG GEBURTS
AG GEBURTST
G GEBURTSTA
 GEBURTSTAG
```

Nachfolgend eine etwas verbesserte Version dieses Programms:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HOME : INPUT "ZEICHENKETTE: ";X$
20 X$ = X$ + " ":X =  LEN (X$)
22  IF X > 22 THEN  PRINT : INVERSE : PRINT "ACHTUNG:": NORMAL
: PRINT : PRINT "ES MUESSEN WENIGER ALS 22 ZEICHEN SEIN": CALL
 - 198: FOR I = 1 TO 2000: NEXT : GOTO 10
25 A = 21 -  INT (X / 2)
26 B = 12 -  INT (X / 2)
30  HOME : VTAB B: HTAB A: PRINT X$
40  FOR I = 1 TO X - 1
50 Y$ =  LEFT$ (X$,1)
60 X$ =  RIGHT$ (X$,X - 1) + Y$
70  HTAB A: PRINT X$
80  NEXT
90  END
```

Dabei wurde berücksichtigt, daß die Anzeige genau so viele Zeilen in Anspruch nimmt wie der erweiterte String X$ Zeichen hat. Für Strings mit mehr als 22 Zeichen wäre die Anzeige nach Beendigung des Programms nicht mehr vollständig, deshalb wurde die Zeile 22 eingefügt, die auch einen entsprechenden Hinweis bei falscher Eingabe liefert. Das Maschinenprogramm CALL-198 erzeugt einen "PIEP"-Ton.
Die Anzeige sollte zudem auf dem Bildschirm zentriert werden. Zu diesem Zweck werden in den Zeilen 25 und 26 die Variablen A (für die horizontale Position) und B (für die vertikale Position) eingeführt.
Mit Hilfe der Länge des Strings aus der Variablen X wird die Anzahl der freien Bildschirmzeilen bzw. Spalten berechnet. In die Zeilen 30 und 70 wurden entsprechende HTAB und VTAB-Befehle eingefügt.

In diesem Programm wird deutlich, wie Stringfunktionen eingesetzt werden können, um Strings zu durchsuchen und zu zerlegen; dieser Vorgang ist auch nötig bei Programmen, die nicht nur Spielerei sind, wie etwa Textverarbeitungsprogramme und Dateiverwaltungsprogramme.
Außerdem sollte deutlich gemacht werden, daß die Bildschirmpositionen nicht notwendigerweise vom Benutzer zahlenmäßig vorgegeben werden müssen, sondern daß der Rechner sie auch berechnen kann - mit den erforderlichen Anweisungen.

7.5.3 Formatierte Bildschirmausgabe

Für Benutzer von Rechnern mit dem Betriebssystem CP/M ist die Formatierung von Bildschirmausgaben kein Problem. Dort gibt es den speziellen Befehl PRINT USING. Dieser steht auf dem APPLE II nicht zur Verfügung, deshalb muß auf etwas umständliche Programmierlösungen für diese Aufgabe zurückgegriffen werden. Dabei ist der Einsatz der Stringfunktionen unbedingt notwendig.

Formatierte Ausgabe bedeutet folgendes:
Man denkt sich den Bildschirm in mehrere senkrechte Streifen aufgeteilt, innerhalb derer Anzeigen erfolgen sollen, und zwar sollen Zahlen so untereinander stehen, daß der Dezimalpunkt sich immer in derselben Bildschirmspalte befindet und eventuell sollen fehlende Nachkommastellen durch Nullen aufgefüllt werden. Sowohl Strings als auch Zahlen werden aber üblicherweise linksbündig geschrieben auf dem Bildschirm.
Das kann durch folgendes Programm verdeutlicht werden, das 2Ø Zahlen in 5 Zeilen und 4 Kolonnen schreibt, deren Anfang durch HTAB festgelegt wird:

```
3   HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4   PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HOME : FOR I = 0 TO 4: FOR J = 0 TO 3: READ X(I,J): NEXT
J,I
20  FOR I = 0 TO 4
30  FOR J = 0 TO 3: HTAB (J * 9 + 1): PRINT X(I,J);: NEXT
40  PRINT : NEXT I
999  END
1000  DATA  12.2345,3,7.8,1789.4
1010  DATA  4,121.789,-23.5,4.121
1020  DATA  1,2,3,4
1030  DATA  65.4386,34218,3.764,9
1040  DATA  0.8,198,34.274,456.1
```

Die Bildschirmanzeige sieht so aus :

```
12.2345   3        7.8       1789.4
4         121.789  -23.5     4.121
1         2        3         4
65.4386   34218    3.764     9
.8        198      34.274    456.1
```

Wie kann man trotzdem etwa folgende Bildschirmausgabe erhalten, die ebenfalls aus 4 Zahlenkolonnen besteht?

```
 12.23        3.00     7.80    1789.40
  4.00      121.79 -  23.50       4.12
  1.00        2.00     3.00       4.00
 65.44   #1234218#     3.76       9.00
  0.80      198.00 -  34.27     456.10
```

Alle Zahlen sind auf 2 Stellen nach dem Komma ausgegeben, und zwar rechtsbündig in vorher festgelegten Feldern. Zudem ist bei Zahlen zwischen -1 und +1 vor dem Dezimalpunkt eine Null eingefügt worden.
Eine Zahl erscheint in " " eingeschlossen, sie war zu lang für das angegebene Feld.

Das entsprechende Programm ist recht umfangreich im Vergleich zu dem einfachen Ausgabeprogramm vorher:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HOME : DIM X(4,3),F$(3)
20  DEF  FN R(X) =  INT (X * 100 + .5) / 100
30  FOR I = 0 TO 4: FOR J = 0 TO 3: READ X(I,J): NEXT J,I
40  FOR J = 0 TO 3: READ F$(J): NEXT
50  FOR I = 0 TO 4
60  FOR J = 0 TO 3:X =  FN R(X(I,J)):X$ =  STR$ (X)
70  IF  LEFT$ (X$,1) <  > "-" THEN X$ = " " + X$
80 L =  LEN (X$): IF  MID$ (X$,2,1) = "." THEN X$ =  LEFT$
(X$,1) + "0" +  RIGHT$ (X$,L - 1):L = L + 1
90  IF L = 2 THEN L = L + 1
100  IF  MID$ (X$,L - 2,1) = "." THEN 130
110  IF  MID$ (X$,L - 1,1) <  > "." THEN X$ = X$ + ".0"
120 X$ = X$ + "0":L =  LEN (X$)
130 F =  LEN (F$(J)): IF L > F THEN X$ = "#" +  MID$ (X$,2,F
- 2) + "#": GOTO 160
140  IF L = F THEN 160
150 X$ =  LEFT$ (X$,1) +  LEFT$ (F$(J),F - L) +  RIGHT$
(X$,L - 1)
160  HTAB (J * 10 + 1): PRINT X$;: NEXT J
170  PRINT : NEXT I
180  END
1000  DATA  12.2345,3,7.8,1789.4
1010  DATA  4,121.789,-23.5,4.121
1020  DATA  1,2,3,4
1030  DATA   65.4386,1234218,3.764,9
1040  DATA   0.8,198,-34.274,456.1
1100  DATA "      ","      ","      ","
"
```

Programmbeschreibung

Sätze	Inhalt
1Ø	Die Arrays X und FØ werden dimensioniert
2Ø	Die Rundungsfunktion R wird definiert
3Ø	Für das Array X werden 5*4 Zahlen eingelesen
4Ø	Die Breite der 4 Bildschirmstreifen wird durch FØ festgelegt in Form von Leerzeichen
5Ø	Beginn der I-Schleife für die Zeilen
6Ø	Beginn der J-Schleife für die Felder Jeweils eine Zahl wird gerundet auf zwei Stellen nach dem Komma und in einen String XØ verwandelt.
7Ø	Falls es sich um eine positive Zahl handelt, erhält der String XØ als erstes Zeichen ein Leerzeichen.
8Ø	Nachdem die Länge des Strings XØ bestimmt wurde und der Variablen L zugewiesen wurde, wird überprüft, ob es sich um eine Zahl zwischen -1 und +1 handelt, die aber ungleich Null ist. Solche Zahlen haben an der zweiten Stelle im String den Dezimalpunkt (Beispiele: "-.3" oder " .3"), weil in Zeile 7Ø die positiven durch ein Leerzeichen ergänzt wurden. Falls eine solche Zahl vorliegt, wird vor den Dezimalpunkt eine Null eingefügt, damit die spätere Ausgabe nicht .8, sondern Ø.8 lautet. Die Länge L muß in diesem Fall um Eins erhöht werden.

Sätze	Inhalt
9Ø	Für einstellige ganze Zahlen ist die Länge 2. In der Zeile 1ØØ würde für den Fall ein Fehler auftreten (L-2=Ø in der MID$-Funktion), also wird L erhöht.
1ØØ-12Ø	Hier erfolgt bei Bedarf das Anfügen von Nullen bei fehlenden Nachkommastellen. 1ØØ Falls die gerundete Zahl genau zwei Nachkommastellen hat, d.h. falls das drittletzte Zeichen von X$ der Dezimalpunkt ist, ist keine Veränderung nötig, die beiden folgenden Zeilen können übersprungen werden. 11Ø Befindet sich weder an der drittletzten noch an der vorletzten Stelle des Strings ein Punkt, handelt es sich um eine ganze Zahl, der String wird um den Punkt und eine Nachkommastelle erweitert. 12Ø Ganze Zahlen bekommen noch eine weitere "Ø" und Zahlen mit nur einer Nachkommastelle ebenfalls eine "Ø" als letztes Zeichen angehängt. Nach diesen Veränderungen muß die Länge des Strings neu bestimmt werden.
13Ø-15Ø	Nun muß X$ in das Feld F$(J) richtig plaziert werden. 13Ø Zunächst muß die Länge F des Feldes bestimmt werden und, falls X$ zu lang ist (L>F), werden das erste und die letzten Zeichen durch " " ersetzt bzw. abgeschnitten. Danach erfolgt der Sprung zur Zeile 16Ø. 14Ø Falls X$ genau das Feld füllt, wird ebenfalls zu Zeile 16Ø gesprungen, es muß keine Veränderung vorgenommen werden.

Sätze	Inhalt
	15Ø X$ wird rechtsbündig in das Feld F$(J) gesetzt. Minuszeichen, sofern sie vorhanden sind, werden an die erste Stelle des Feldes gesetzt, der Zwischenraum besteht aus den bisher schon in F$(J) vorhandenen Leerzeichen.
16Ø	Nun kann nach all diesen Vorbereitungen endlich der Ausdruck erfolgen; wieder wird durch HTAB der Beginn der Felder festgelegt.
17Ø	Nach einem Zeilenvorschub (wegen des Semikolon hinter PRINT X$ in Zeile 16Ø) wird die I-Schleife für die Zeile geschlossen.
18Ø	Ende des Programms
1ØØØ-11ØØ	DATA-Statements für die READ-Befehle der Zeilen 3Ø und 4Ø

Man muß schon etwas Zeit aufwenden, um diese formatierte Bildschirmausgabe zu programmieren. Es wäre deshalb sinnvoll, zumindest Teile dieses Programms oder eines ähnlichen (es gibt noch mehr Möglichkeiten, die Formatierung des Bildschirms zu programmieren) immer als Unterprogramm bereitzuhalten, das dann in das jeweilige Hauptprogramm eingemischt werden kann, natürlich mit variablem Endwert für die I- bzw. J-Schleife. Ob man die EXEC-Methoden wählt oder APA ist dabei unwesentlich.

7.5.4 Bildschirmmasken

Unter der Bezeichnung Bildschirmmasken versteht man meist die Aufteilung des Bildschirms für die Eingabe von Werten (Zahlen und Texten), die einer Datei zugewiesen werden sollen oder sofort im weiteren Programm verarbeitet werden. Dabei ist es wichtig, den Benutzer so zu führen, daß er weiß, was eingegeben werden soll und in welcher Form. Eine einfache Form einer Bildschirmmaskenerstellung zeigt das folgende Programm:

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  HOME : HTAB 6: PRINT "P E R S O N A L A N G A B E N": PRINT
: FOR Q = 1 TO 40: PRINT "-";: NEXT : PRINT : POKE 34,4: HOME

20  PRINT "NAME: ..............";: HTAB 22: PRINT " VORNAME:
........."
30  VTAB 8: FOR Q = 1 TO 34: PRINT ".";: NEXT : HTAB 36: PRINT
"...."
40  HTAB 14: PRINT "STRASSE";: HTAB 37: PRINT "NR."
50  VTAB 12: PRINT "......";: HTAB 9: FOR Q = 1 TO 20: PRINT
".";: NEXT : PRINT
60  PRINT " PLZ";: HTAB 15: PRINT "WOHNORT"
70  VTAB 16: PRINT ".....";: HTAB 11: PRINT ".........";:
HTAB 28: PRINT "........."
75  PRINT "ALTER";: HTAB 11: PRINT "STEUERKLASSE";: HTAB 28:
PRINT "KINDERZAHL"
80  PRINT : HTAB 15: PRINT "KINDERZAHL  : ...."
90  VTAB 19: PRINT "MACHEN SIE IHRE EINGABE DORT, WO DAS": PRINT
"LICHTZEICHEN BLINKT NUR BIS ZUR VORGE-": PRINT "GEBENEN LAENGE.":
PRINT "DRUECKEN SIE DANN DIE <RETURN> - TASTE."
100 PV = 5:PH = 7: GOSUB 1000:N$ = X$:PH = 32: GOSUB 1000:VN$
= X$
110 PV = 8:PH = 1: GOSUB 1000:S$ = X$:PH = 36: GOSUB 1000:NR$
= S$
120 PV = 12:PH = 1: GOSUB 1000:PL$ = X$:PH = 9: GOSUB 1000:W$
= X$
130 PV = 16:PH = 1: GOSUB 1000:A$ = X$:PH = 16: GOSUB 1000:SK$
= X$:PH = 32: GOSUB 1000:KZ$ = X$
140  VTAB 19: CALL  - 958: HTAB 6: INVERSE : PRINT "DRUECKEN
SIE IRGENDEINE TASTE": NORMAL : POKE  - 16368,0: WAIT  - 16384,128:
POKE  - 16368,0: HOME
150  END
1000 X$ = "": VTAB PV
1010  HTAB PH: GET E$: PRINT E$;: IF E$ <  >  CHR$ (13) THEN
PH = PH + 1:X$ = X$ + E$: GOTO 1010
1020  RETURN
```

Programmbeschreibung

Sätze	Inhalt
1Ø	Erzeugen einer Überschrift, Sperren von 4 Bildschirmzeilen am oberen Bildschirmrand durch POKE 34,4.
2Ø- 8Ø	Kennzeichnen der Eingabefelder und der geforderten Eingaben.
9Ø	Hinweis auf die Bedienung.
1ØØ-13Ø	Eingabe der Werte. Nach der Festlegung der vertikalen und horizontalen Position wird das Unterprogramm 1ØØØ jeweils angesprungen, danach der Wert der Variablen X$ des Unterprogramms an die jeweilige Hauptprogrammvariable übergeben.
14Ø	Programmunterbrechung durch WAIT.
15Ø	Im Beispiel ist hier das Programm zu Ende, ansonsten könnten ab 15Ø weitere Anweisungen zur Verarbeitung der soeben eingegebenen Daten folgen.
1ØØØ	Unterprogramm für die Eingabe. Es werden zunächst der Ergebnisstring X$ gleich dem leeren String gesetzt und die vertikale Position bestimmt. PV muß aus dem Hauptprogramm übergeben werden, ebenso PH in der nächsten Zeile.
1Ø1Ø	Durch GET wird jeweils 1 Zeichen eingelesen und durch PRINT ausgegeben. Solange nicht RETURN gedrückt wird, werden die eingegebenen Zeichen dem String X$ hinzugefügt und die horizontale Position jeweils um Eins erhöht.
1Ø2Ø	Rücksprung ins Hauptprogramm. Er erfolgt, sobald E$=CHR$(13), d.h. sobald die RETURN -Taste gedrückt wird.

In diesem Programm fehlen vor allem die Einrichtungen, um Fehler von Benutzern abzufangen. So wird der Benutzer zwar angewiesen, die Eingaben nur bis zur vorgegebenen Länge zu machen, aber es gibt keine Befehle im Programm, die ihn daran hindern, falls er sich nicht an die Anweisungen hält. Außerdem werden die Eingaben nicht auf Plausibilität geprüft.
Da für alle Eingaben Stringvariablen verwendet werden, werden Buchstabenfolgen für Alter oder Hausnummer ebenfalls akzeptiert. Dies müßte noch verbessert werden.

Mehr Programmieraufwand erfordert zunächst der nachfolgend gelistete "Maskengenerator"; Das Programm benötigt immerhin mehr als 3 Seiten Listing. Es erstellt eine EXEC-Datei mit dem Namen MASKE, die die Programmzeilen für eine Bildschirmmaske enthält. Der Vorteil liegt zum einen in der vielseitigen Benutzung, da die Zeilennummern beliebig gewählt werden können, also an der gewünschten Stelle ins Hauptprogramm eingefügt werden können durch den Befehl EXEC. Ein ganz wesentlicher Vorteil liegt aber vor allem darin, daß man unmittelbar den Bildschirm zunächst einmal so beschreiben kann, wie man ihn beim Programmlauf sehen will, d.h. ohne PRINT-Befehle und ohne das lästige Zählen.
Man muß nur die Eingabefelder durch ein spezielles Zeichen markieren und die gewünschte Bezeichnung davorsetzen. Über die Speicherstellen 1Ø48 bis 2Ø47 wird der so gestaltete Bildschirminhalt dann in die erzeugten Programmzeilen eingesetzt.

Allerdings hat dieser Maskengenerator einige kleinere Mängel. Es werden z.B. für die Eingabefelder auch Variablennamen erfragt, die nachfolgenden Benutzereingaben werden aber nicht auf Syntaxfehler geprüft. Dies führt dann nach dem Aufruf des erstellten Programms MASKE zu Syntaxfehlern, d.h. es handelt sich nicht um eine abbruchsichere Maske.

Das Programm MASKENGENERATOR stammt von Frank Brall, es wurde hier weitgehend in der Originalfassung übernommen. Die geringfügigen Änderungen betrafen nur offensichtliche Tippfehler. Ablauf oder Syntax wurden nicht verändert. Das Listing wurde der Zeitschrift COMPUTRONIC, April 84, entnommen.

Das Programm erklärt seine Bedienung bei Start selbst, auf eine Programmbeschreibung haben wir verzichtet, da es sich um ein Fremdprogramm handelt.
Mit diesem sehr umfangreichen Beispiel soll der Abschnitt Bildschirmgestaltung abgeschlossen werden.

```
80  ONERR  GOTO 1360
90  HOME
100  PRINT "BILDSCHIRMMASKENGENERATOR"
110  PRINT : PRINT "      (C) BEI FRANK BRALL"
120  PRINT : PRINT
130  PRINT "DIESES PROGRAMM ERSTELLT EINE ABBRUCH-": PRINT
"SICHERE EINGABE-MASKE, DIE IN BASIC": PRINT "GESCHRIEBEN
IST !"
170  PRINT : PRINT "NACH TASTENDRUCK ERSCHEINT EIN BEISPIEL,":
PRINT "WELCHES MIT DEM EDITOR ERSTELLT WURDE."
180  GET E$: IF E$ = "" THEN 180
190  HOME : PRINT "KARTEI-NR.: @@@       DATUM: @@@": PRINT
"----------------------------------------": PRINT : PRINT
"NAME: @@@@@@@@@@@@  VORNAME: @@@@@@@@@@@@": PRINT "LETZTER
WOHNORT:@@@@@@@@@@@@@@@@@@@@@@@@@@ "
200  PRINT : PRINT : PRINT "HEUTIGER WOHNORT:  PLZ @@@@
@@@@@@@@@@@@ ": PRINT : PRINT : PRINT "SONSTIGE BEMERKUNGEN:
@@@@@@@@@@@@@@@@@@@@": PRINT : PRINT "@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@"
210  GET E$: IF E$ = "" THEN 210
220  HOME
310  DIM X(100),Y(100),V$(100),L(100)
320  FOR I = 768 TO 880: READ E: POKE I,E: NEXT I
330  DATA  164,36,169,32,145,40,173,16,192,173,0,192,16,251,72,164,36,
177,40,73,128,145,40,104,201,133,208,1,96,201,131,208,6,32,88,252,76,0
,3,201,141,240,10,201,136,240,12,166,36,224,39,208,6
340  DATA  166,37,224,23,240,43,201,149,208,16,166,36,224,39,208,5,169
,141,76,79,3,230,36,76,102,3,201,129,208,6,32,26,252,76,102,3,201,154,
208,6,32,102,252,76,102,3,32,237,253,164,36,177,40,73,128,145,40,76,6,
3
350  DATA  1024,1152,1280,1408,1536,1664,1792,1920,1064,1192,1320,1448
,1576,1704,1832,1960,1104,1232,1360,1488,1616,1744,1872,2000
360  INPUT "ERSTE ZEILEN-NR. DER MASKE ? ";ZN: HOME
370  PRINT "MASKENEDITOR": PRINT "-----------------------------------
---"
380  PRINT : PRINT : PRINT "CRTL-A   =  CURSOR 1 ZEILE
HOEHER"
390  PRINT : PRINT "CRTL-Z   =  CURSOR 1 ZEILE TIEFER"
400  PRINT : PRINT "  ->     =  CURSOR 1 STELLE WEITER"
410  PRINT : PRINT "  <-     =  CURSOR 1 STELLE ZURUECK"
420  PRINT : PRINT "RETURN   =  ZUM NAECHSTEN ZEILENANFANG"
430  PRINT : PRINT "CTRL-C   =  SCHIRM LOESCHEN"
440  PRINT : PRINT "CTRL-E   =  MASKE ERSTELLEN"
450  PRINT : PRINT "  @      =  MARKIERUNGSZEICHEN FUER"
460  PRINT : PRINT "            FENSTER;BELIEBIG ANREIHBAR"
470  PRINT : CALL 768: GET E$
```

```
 POKE 36,0: POKE 37,0: PRINT
)$ =  CHR$ (4): PRINT D$;"DELETE MASKE"
 CALL 1002
 PRINT D$;"OPEN MASKE"
 PRINT D$;"WRITE MASKE"
 PRINT ZN;"REM ¥¥¥¥ BILDSCHIRMMASKE ¥¥¥":ZN = ZN +

 PRINT ZN;"GOTO ";ZN + 21:ZN = ZN + 1:NR = ZN
 PRINT ZN;"VTAB(TY):HTAB(TX):INVERSE:FOR ZE=1 TO LE:?CHR$(32);:NEX
":ZN = ZN + 1
 PRINT ZN:"VTAB(TY):HTAB(TX):?LEFT$(ME$,LE);":ZN =
 1
 PRINT ZN;"VTAB(TY):HTAB(TX):GE$="; CHR$ (34); CHR$
;":ZE=0":ZN = ZN + 1
 PRINT ZN;"GET EI$:IF EI$<>CHR$(13)ANDEI$<>CHR$(21)
";ZN + 3:ZN = ZN + 1
 PRINT ZN;"IF EI$=CHR$(21) THEN ";ZN - 1:ZN = ZN +

 PRINT ZN;"GE$=LEFT$(ME$,LE):GOTO";ZN + 13:ZN = ZN

 PRINT ZN;"FOR ZE=1 TO LE:?CHR$(32);:NEXTZE:VTAB(TY):HTAB(TX):ZE=0
O";ZN + 2:ZN = ZN + 1
 PRINT ZN;"GET EI$":ZN = ZN + 1
 PRINT ZN;"IF EI$=CHR$(27) THEN GE$=ME$:GOTO";ZN =
 1
 PRINT ZN;"IF EI$=CHR$(13) THEN";ZN + 8:ZN = ZN + 1
 PRINT ZN;"IF EI$=CHR$(21) THEN";ZN - 3:ZN = ZN + 1
 PRINT ZN;"IF EI$=CHR$(8) AND ZE=0 THEN";ZN - 4:ZN
| + 1
 PRINT ZN;"IF EI$=CHR$(8) THEN ZE=ZE-1:GOTO";ZN + 3:ZN
| + 1
 PRINT ZN;"IF ZE=LE-1 THEN GE$=GE$+EI$:ZE=ZE+1:GOTO";ZN
ZN = ZN + 1
 PRINT ZN;"ZE=ZE+1":ZN = ZN + 1: PRINT ZN;"? EI$;:GE$=GE$+EI$":ZN
+ 1
 PRINT ZN;"GOTO";ZN - 9:ZN = ZN + 1
 PRINT ZN;"IF GE$<>"; CHR$ (34); CHR$ (34)" THEN GE$=LEFT$(GE$,ZE)
= ZN + 1
 PRINT ZN;"NORMAL:VTAB(TY):HTAB(TX):FOR ZE=1 TO LE:?CHR$(32);:NEXT
':ZN = ZN + 1
 PRINT ZN;"VTAB(TY):HTAB(TX):?GE$;:RETURN":ZN = ZN

 PRINT ZN;"HOME":ZN = ZN + 1
 FOR U = 1 TO 24
 READ A1:ST$ = ""
 FOR I = A1 TO A1 + 39:C =  PEEK (I) - 128
 IF C = 64 AND F = 0 THEN F = 1:X(Z) = I + 1 - A1:Y(Z)

 IF C < > 64 AND F = 1 THEN F = 0:L(Z) = I + 1 - A1
(Z):Z = Z + 1
 IF C = 64 OR C = 34 THEN C = 32
ST$ = ST$ +  CHR$ (C)
 NEXT I
```

```
830  IF F = 1 THEN F = 0:L(Z) = I + 1 - A1 - X(Z):Z = Z
+ 1
840 I = 0:T = 40
850  IF  MID$ (ST$,T,1) <  > " " THEN 870
860 I = I + 1:T = T - 1: IF T > 0 THEN 850
870  IF U = 24 AND I < 2 THEN I = 2
880  IF U = 24 AND I = 40 THEN  PRINT ZN;"PRINT"; CHR$
(34); CHR$ (34);";": GOTO 930
890  IF I = 40 THEN  PRINT ZN;"PRINT": GOTO 930
900  IF I = 0 THEN  PRINT ZN;"PRINT"; CHR$ (34);ST$; CHR$
(34);";": GOTO 930
910  IF U = 24 THEN  PRINT ZN;"PRINT"; CHR$ (34);: PRINT
 LEFT$ (ST$,41 - I);: PRINT  CHR$ (34);";": GOTO 930
920  PRINT ZN;"PRINT"; CHR$ (34);: PRINT  LEFT$ (ST$,40
- I);: PRINT  CHR$ (34)
930 ZN = ZN + 1: NEXT U
940  PRINT D$;"CLOSE MASKE"
950  IF Z = 0 THEN  HOME : PRINT "FEHLER !!": PRINT : PRINT
"ES FEHLT DIE KENNZEICHNUNG DER FENSTER !": PRINT : PRINT
: PRINT "BEISPIEL:          DATUM : @@@@@@@ ": PRINT : PRINT
"                NAME @@@@@@@@@@@@    NR. @@": END
960  VTAB 24: HTAB 1: PRINT "
                ";:U = 24
970  FOR Q = 0 TO Z - 1
980  IF U = 24 AND Y(Q) = 24 THEN U = 22: VTAB 24: HTAB
1: PRINT  LEFT$ (ST$,39);
990  FLASH
1000  VTAB (Y(Q)): HTAB (X(Q)): PRINT "@";
1010  NORMAL
1020  VTAB (U): HTAB 1: PRINT "VARIABLENNAME ";Q + 1;"
?        ";
1030 G$ = "": HTAB 18
1040  GET E$
1050  IF E$ =  CHR$ (8) THEN 1020
1060  IF E$ <  >  CHR$ (13) THEN  PRINT E$;:G$ = G$ + E$:
GOTO 1040
1070 V$(Q) = G$: IF G$ =  CHR$ (13) THEN 1020
1080  HTAB (X(Q)): VTAB (Y(Q)): PRINT "@";
1090  NEXT Q
1100  HOME : PRINT D$;"MON O"
1110  PRINT D$;"APPEND MASKE"
1120  PRINT D$;"WRITE MASKE"
1130  FOR Q = 0 TO Z - 1
1140 F = 0
1150  FOR I = 1 TO  LEN (V$(Q)): IF  MID$ (V$(Q),I,1) =
"$" THEN F = 1
1160  NEXT I
1170  IF F = 1 THEN  PRINT ZN;"VTAB(";Y(Q);"):HTAB(";X(Q);"):?LEFT$(";
V$(Q);",";L(Q);");"
1180  IF F = 0 THEN  PRINT ZN;"VTAB(";Y(Q);"):HTAB(";X(Q);"):ME$=STR$(
";V$(Q);"):?RIGHT$(ME$,";L(Q);");"
1190 ZN = ZN + 1: NEXT Q:F = 0
```

```
1200  FOR Q = 0 TO Z - 1
1210  PRINT ZN;"TX=";X(Q);":TY=";Y(Q);":LE=";L(Q):ZN =
ZN + 1
1220  FOR I = 1 TO  LEN (V$(Q)): IF  MID$ (V$(Q),I,1) =
"$" THEN F = 1
1230  NEXT I
1240  IF F = 1 THEN  PRINT ZN;"ME$=";V$(Q);":GOSUB";NR:ZN
= ZN + 1
1250  IF F = 1 AND Q > 0 THEN  PRINT ZN;"IF EI$=CHR$(27)
THEN";ZN - 6:ZN = ZN + 1
1260  IF F = 1 THEN  PRINT ZN;V$(Q);"=GE$":ZN = ZN + 1
1270  IF F = 0 THEN  PRINT ZN;"ME$=RIGHT$(STR$(";V$(Q);"),";L(Q);"):GO
SUB";NR;":IF GE$=";CHR$ (34);CHR$ (34);" THEN";ZN:ZN
= ZN + 1
1280  IF F = 0 AND Q > 0 THEN  PRINT ZN;"IF EI$=CHR$(27)
THEN";ZN - 6:ZN = ZN + 1
1290  IF F = 0 THEN  PRINT ZN;V$(Q);"=VAL(GE$)":ZN = ZN
+ 1
1300 F = 0: NEXT Q
1310  PRINT ZN;"REM * ENDE DER MASKE * "
1320  PRINT D$;"CLOSE MASKE"
1330  HOME : PRINT "DIE MASKE IST ERSTELLT !": PRINT :
PRINT : PRINT "DURCH DEN BEFEHL ";
1340  INVERSE : PRINT "EXEC MASKE": NORMAL : PRINT "WIRD
SIE IN DEN SPEICHER GESCHRIEBEN."
1350  END
1360  REM   *** FEHLERKONTROLLE ***
1370 QE =  PEEK (222)
1380  IF QE = 6 THEN 500
1390  PRINT  CHR$ (7): PRINT D$;"CLOSE MASKE"
1400  PRINT : INVERSE : PRINT "FEHLERCODE ";QE: NORMAL
: END
```

7.6 Sortieren

Im Kapitel 1 wurde ein Beispielprogramm für den Sortiervorgang ausführlich erläutert. Es handelt sich dabei um einen sogenannten Bubble-Sort, es werden alle Elemente eines Arrays mit allen nachfolgenden verglichen und bei falscher Reihenfolge wird vertauscht.
Das Programm erfordert zwei geschachtelte Schleifen, der Umfang der inneren Schleife hängt vom Index der äußeren Schleife ab. Dieses Verfahren ist vergleichsweise einfach zu programmieren und es ist sicher.
Andererseits ist es aber auch ziemlich langsam, da für ein Array vom Umfang N genau N(N-1)/2 Vergleiche durchgeführt werden müssen. Für N=1ØØ sind dies schon 495Ø Vergleiche. Genaue Ausführungszeiten lassen sich nicht angeben, da diese nicht nur von der Anzahl der Vergleiche, sondern auch von der Anzahl der erforderlichen Vertauschungen abhängen, also davon, ob die Liste schon teilweise sortiert ist, eventuell umgekehrt sortiert oder völlig unsortiert.
Diese relativ lange Ausführungszeit war sicherlich der Grund für eine große Menge von Veröffentlichungen von Sortierroutinen. Durch umfangreiche Testläufe hat sich gezeigt, daß es keinen "besten" Sortieralgorithmus gibt. Einige Routinen, die für Großrechner entwickelt wurden und dort hervorragende Ausführungszeiten erreichen, stellen sich für Kleinrechner als ungeeignet heraus, sind auf Kleinrechner möglicherweise sogar langsamer als die Methode der direkten Vergleiche.
Da Ausführungszeiten auch bestimmt werden vom Betriebssystem, ist dasselbe Programm nicht auf allen Kleinrechnern gleich gut.

Man teilt die Sortierroutinen nach der Problemstellung in 2 Gruppen ein:

X-Sorter und X/Y-Sorter.

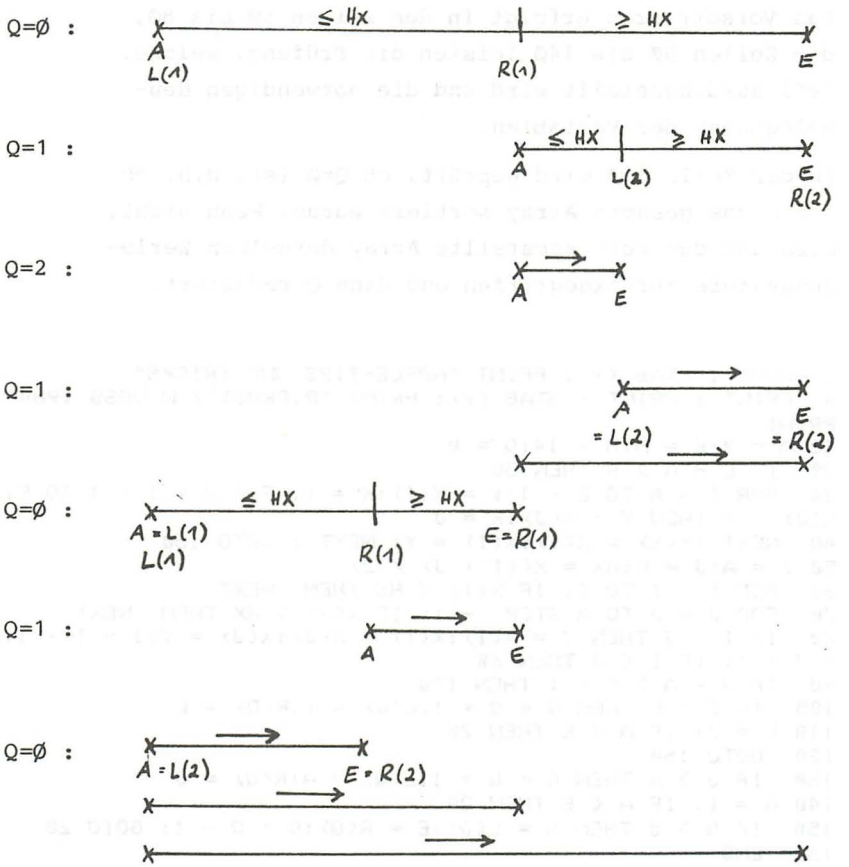
X-Sorter sind solche Programme, die ein Array (nämlich X) der Größe nach - aufwärts oder abwärts - sortieren. X/Y-Sorter hingegen sortieren ein Array (nämlich X) und führen dabei jeweils die entsprechenden Werte eines zweiten Arrays (Y) mit.
Für den APPLE als besonders günstig erwiesen haben sich für die erste Aufgabe der sogenannte Super-Quicksort von KOLBE und für die zweite Aufgabe ein X/Y-Sorter von VELLEMAN und HOAGLIN.
Das Programm für die X-Sortierroutine ist recht kompliziert aufgebaut, basiert aber auf einer einfachen Idee, die man auch im Alltag immer wieder bei Sortierarbeiten einsetzt, die nicht mit dem Computer erledigt werden, z.B. bei der Ablage von Schriftstücken nach dem Alphabet oder nach Sachgebieten. Es ist zeitsparend, zunächst grob vorzusortieren und dann erst - möglicherweise in mehreren Schritten - die "Feinsortierung" vorzunehmen; zum Beispiel werden erst alle Schriftstücke nach dem entsprechenden Anfangsbuchstaben geordnet und erst in einem zweiten Schritt werden dann alle Schriftstücke desselben Buchstabens sortiert.
Ähnlich arbeitet die Routine nach dem Algorithmus von VELLEMAN und HOAGLIN. Man sortiert zunächst das gesamte Array so, daß die Werte des ersten Teils alle kleiner oder gleich einem Vergleichswert (HX) sind und die Werte des zweiten Teils größer oder gleich diesem Wert.

Als Vergleichswert dient der Arraywert an der mittleren Position, d.h. X(N/Z). Falls N/Z nicht ganzzahlig ist, nimmt der Rechner automatisch den nächstkleineren ganzzahligen Wert als Index an. Das "längere" Stück wird dann zunächst zurückgestellt, mit Hilfe der Variablen L(Q) und R(Q) werden Anfangs- und Endindex dieses Teils gespeichert. Dabei gibt Q die Zerlegungsstufe an. Nun werden A oder E neu belegt.

Das "kürzere" Stück wird weiter bearbeitet, d.h. das mittlere Element dieses Teilarrays wird zum neuen Vergleichswert HX und wieder werden die Elemente gruppiert in solche, die kleiner oder gleich HX sind und solche, die größer oder gleich HX sind. Q wird um Eins erhöht, das größere Teilstück zurückgestellt, das kleinere weiter untersucht.

Dies geschieht solange, bis das kleinere Teilstück noch höchstens 15 Elemente enthält. Diese werden dann sortiert (Zeilen 3Ø und 4Ø).
Danach das zurückgestellte Stück dieser Zerlegungsstufe; durch Q=Q-1 geht man eine Stufe zurück. Falls dieses Stück länger ist als 15 Elemente, wird wieder vorsortiert, ansonsten kann die Feinsortierung durch die Zeilen 3Ø und 4Ø erfolgen. Die jeweiligen sortierten Teile einer Stufe passen "nahtlos" aneinander, da sie durch denselben Vergleichswert auseinandergeschnitten wurden.

Zu diesem Vorgehen eine schematische Darstellung:



Dabei bedeutet ein Pfeil, daß der entsprechende Arraybereich sortiert ist.

Das Vorsortieren erfolgt in den Zeilen 5Ø bis 8Ø, die Zeilen 9Ø bis 14Ø leisten die Prüfung, welcher Teil zurückgestellt wird und die notwendigen Neubelegungen der Variablen.

In der Zeile 15Ø wird geprüft, ob Q=Ø ist, d.h. ob schon das gesamte Array sortiert wurde. Wenn nicht, wird auf das zurückgestellte Array derselben Zerlegungsstufe zurückgegriffen und dann Q reduziert.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10 A = 0:E = N:H = 14:Q = 0
20  IF E - A > H THEN 50
30  FOR I = A TO E - 1:Y = X(I):K = I: FOR J = I + 1 TO E: IF
X(J) < Y THEN Y = X(J):K = J
40  NEXT :X(K) = X(I):X(I) = Y: NEXT : GOTO 150
50 I = A:J = E:HX = X((I + J) / 2)
60  FOR I = I TO E: IF X(I) < HX THEN  NEXT
70  FOR J = J TO A STEP  - 1: IF X(J) > HX THEN  NEXT
80  IF I < J THEN Y = X(I):X(I) = X(J):X(J) = Y:I = I + 1:J
= J - 1: IF I < J THEN 60
90  IF J - A > E - I THEN 130
100  IF I < E THEN Q = Q + 1:L(Q) = I:R(Q) = E
110 E = J: IF A < E THEN 20
120  GOTO 150
130  IF J > A THEN Q = Q + 1:L(Q) = A:R(Q) = J
140 A = I: IF A < E THEN 20
150  IF Q > 0 THEN A = L(Q):E = R(Q):Q = Q - 1: GOTO 20
160  END
```

Der X/Y-Sorter arbeitet unkomplizierter, allerdings auch langsamer.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10 L = N - 1
20 L =  INT ((L - 2) / 3) + 1: FOR I = 1 TO N - L:J = I + L:HX
= X(J):HY = Y(J): IF X(I) <  = HX THEN 70
30 Q = I
40 X(J) = X(Q):Y(J) = Y(Q):J = Q: IF Q <  = L THEN 60
50 Q = Q - L: IF X(Q) > HX THEN 40
60 X(J) = HX:Y(J) = HY
70  NEXT I: IF L > 1 THEN 20
80  END
```

Das ursprüngliche X-Array wird "gedrittelt" (wobei der mittlere Teil eventuell etwas länger ist, wenn N nicht durch 3 teilbar ist).

Nun wird das erste Element des ersten Teils mit dem ersten Element des zweiten Teils verglichen und gegebenenfalls vertauscht, dann das zweite Element des ersten Teils mit dem zweiten des zweiten Teils usw.
Der Abstand zwischen den verglichenen Elementen beträgt immer N. Für jedes I aus dem ersten Bereich gilt nach diesem ersten Durchlauf: X(I)<=X(I+L).

Nun werden die Elemente aus dem zweiten Teil mit den entsprechenden aus dem dritten Teil verglichen, also wieder mit den Elementen im Abstand L. Das Größere von beiden gehört ins letzte Drittel. Das Kleinere von beiden wird mit dem entsprechenden des ersten Drittels verglichen und gegebenenfalls dort plaziert. Nach diesem Durchlauf gilt für alle I aus dem ersten Teil, d.h. für alle I<=L:

X(I)<=X(I+L)<=X(I+2L).

Nun wird L verkleinert, und zwar zu einem Drittel der nächsten durch 3 teilbaren Zahl. An einem Beispiel wird vermutlich deutlicher, was gemeint ist:

Für N=2ØØ wird zunächst L auf 199 gesetzt (Zeile 1Ø), danach wird L berechnet zu
L=INT((199-2)/3)+1=INT(197/3)+1=INT(65.666)+1=65+1=66

Dies ist aber 198/3 und 198 ist die durch 3 teilbare Zahl, die 199 am nächsten liegt. Jede ganze Zahl hat als benachbarte Zahl eine, die ein Vielfaches von 3 ist, oder sie ist selbst durch 3 teilbar.

Für den ersten Durchlauf der Schleife in den Zeilen 2Ø bis 7Ø hat L für N=2ØØ also den Wert 66.
Danach wird es entsprechend der Formel in Zeile 2Ø auf etwa ein Drittel reduziert. Da 66 durch 3 teilbar ist, wird L genau 22; das Array ist zerlegt in 9 Teile. Nun wird derselbe Prozeß durchgeführt.
Jedes Element eines beliebigen Teils wird mit den entsprechenden Elementen aller davorliegenden Teile verglichen und gegebenenfalls verschoben. Nach dem ersten Durchlauf der Schleife hatte man 66 geordnete Reihen der Länge 3, die ineinandergeschoben waren.
Nach dem zweiten Durchlauf hat man 22 geordnete Reihen der Länge 9, die ineinandergeschoben sind.

Die nächste durch 3 teilbare Zahl bei 22 ist 21, L wird also auf 7 reduziert; es entstehen 28 Teilstücke. Von 7 sinkt L auf 2; jedes Element wird also mit dem übernächsten verglichen. Nach diesem Durchlauf hat man nur noch 2 ineinandergeschobene geordnete Reihen.
Nachdem L auf 1 reduziert wurde, wird die Schleife noch einmal durchlaufen - jetzt werden benachbarte Elemente verglichen - und das Ergebnis ist die geordnete Reihe.

In der vorhergehenden Erläuterung wurde immer nur von einem Array gesprochen. Dies ist auch richtig, da das Y-Array immer nur mitgeführt wird, seine Elemente also nur dann einbezogen werden, wenn Werte des X-Arrays verschoben werden, also in den Zeilen 4Ø und 6Ø.

Nach diesen etwas theoretischen Erklärungen soll nun ein Anwendungsbeispiel diesen Abschnitt beschließen. Die Routinen wurden zwar für Zahlen entwickelt, können aber auch auf Stringvariablen übertragen werden. Es soll nämlich ein Programm zur Erstellung eines Registers vorgestellt werden, eines Stichwortverzeichnisses mit den Seitenzahlen, auf denen die Stichworte nachzuschlagen sind. Die Wörter müssen alphabetisch sortiert werden und dabei müssen die Seitenzahlen mitgeführt werden. Danach sollen alle Seitenzahlen desselben Stichwortes der Größe nach sortiert werden.

Beide Routinen werden eingesetzt, natürlich in abgeänderter Form. Im X/Y-Sorter wird X durch X$ ersetzt, im X-Sorter X durch Y. Dies kann etwa durch das Programm REPLACE aus dem APPLE-Doc erfolgen.
Sie werden dann als Subroutinen eingebaut. Es soll darauf hingewiesen werden, daß durch den Übergang zu Stringvariablen die Optimalität des X/Y-Sorters möglicherweise verlorengeht.

```
3  HOME : HTAB (9): PRINT "APPLE-TIPS UND TRICKS"
4  PRINT : PRINT : HTAB (9): PRINT "R.PRUST / W.VOSS 1984":
PRINT
10  REM  ***  ERSTELLUNG EINES REGISTERS ***
20 N = 27
30  HOME : TEXT : DIM X$(N),Y(N)
40  FOR I = 1 TO N: READ X$(I),Y(I): NEXT
50  GOSUB 1000
60 Z = 1: FOR M = Z TO N - 1: IF X$(M) = X$(M + 1) THEN  NEXT
M
70  IF M > Z THEN A = Z:E = M: GOSUB 2000
80  PRINT X$(Z);: HTAB 20: PRINT Y(Z);: IF Z = M THEN 130
90  FOR Q = Z + 1 TO M: IF Y(Q) = Y(Q - 1) THEN 120
100  PRINT ", ";: IF  PEEK (36) > 36 THEN  PRINT : HTAB 20
110  PRINT Y(Q);
120  NEXT Q
130  PRINT : IF M = N THEN  END
140  IF M = N - 1 THEN  PRINT X$(N);: HTAB 20: PRINT Y(N): END

150 Z = M + 1: NEXT M
160  END
1000 L = N - 1
1010 L =  INT ((L - 2) / 3) + 1: FOR I = 1 TO N - L:J = I +
L:HX$ = X$(J):HY = Y(J): IF X$(I) <  = HX$ THEN 1060
1020 Q = I
1030 X$(J) = X$(Q):Y(J) = Y(Q):J = Q: IF Q <  = L THEN 1050
1040 Q = Q - L: IF X$(Q) > HX$ THEN 1030
1050 X$(J) = HX$:Y(J) = HY
1060  NEXT I: IF L > 1 THEN 1010
1070  RETURN
2000 H = 14:Q = 0
2010  IF E - A > H THEN 2040
2020  FOR I = A TO E - 1:Y = Y(I):K = I: FOR J = I + 1 TO E:
IF Y(J) < Y THEN Y = Y(J):K = J
2030  NEXT :Y(K) = Y(I):Y(I) = Y: NEXT : GOTO 2140
2040 I = A:J = E:HY = Y((I + J) / 2)
2050  FOR I = I TO E: IF Y(I) < HY THEN  NEXT
2060  FOR J = J TO A STEP  - 1: IF Y(J) > HY THEN  NEXT
2070  IF I < J THEN Y = Y(I):Y(I) = Y(J):Y(J) = Y:I = I + 1:J
= J - 1: IF I < J THEN 2050
2080  IF J - A > E - I THEN 2120
2090  IF I < E THEN Q = Q + 1:L(Q) = I:R(Q) = E
2100 E = J: IF A < E THEN 2010
2110  GOTO 2140
2120  IF J > A THEN Q = Q + 1:L(Q) = A:R(Q) = J
2130 A = I: IF A < E THEN 2010
2140  IF Q > 0 THEN A = L(Q):E = R(Q):Q = Q - 1: GOTO 2010
2150  RETURN
3000  DATA      GOTO,13, REM,24, INDEX,234, GOTO,52, DIMENSION,53,
INPUT,12, GOTO,4, PRINT,12, PRINT,78, INPUT,90, GOTO,4
3010  DATA      LEFT$,198, MID$,286, HTAB,22, PRINT,22, GOTO,56,
VTAB,2, IF..THEN,34, ARRAY,92
3020  DATA     ARRAY,76, REM,15, INPUT,57, INDEX,112, DIM,160
3030  DATA     GOTO,234, GOTO,143, VTAB,28
```

Kapitel 8: Alternativen zur BASIC-Programmierung

8.1 Vorbemerkungen

Alle vorangegangenen Kapitel hatten mit der Frage zu tun, inwieweit mit geschickten BASIC-Programmen und unter Ausnutzung der Möglichkeiten, die der APPLE II bietet, bestimmte Probleme durch den Rechnereinsatz gelöst werden können.

Es hat sich dabei gezeigt, daß diese Möglichkeiten schon so weit ausgebaut sind, daß auch kompliziertere Problemstellungen durchaus erfolgreich bewältigt werden.

Gleichwohl darf nicht übersehen werden, daß die Programmiersprache BASIC eine relativ junge Entwicklung ist und deshalb bei weitem noch nicht all das bieten kann, was beispielsweise die "alten", traditionellen Programmiersprachen an Problemlösungskapazitäten bereitstellen können.

Folgende Überlegung tritt noch hinzu: Nicht für alle Probleme, vor denen man steht, muß man selbst Lösungen finden, weil viele - vor allem häufig wiederkehrende Fragestellungen - längst bearbeitet wurden. Man kann deshalb relativ häufig auf schon fertige Programme oder Programmsysteme zurückgreifen, die als Software-Angebote bereitstehen.

Mit beiden Aspekten, also mit traditionellen Programmiersprachen und mit Softwareprogrammen beschäftigt sich übersichtsweise dieses Kapitel.
Dabei geht es jetzt nicht mehr darum, Programme oder einzelne Programmsegmente vorzustellen, sondern es geht eher um eine generelle Übersicht.
Wir können nämlich nicht davon ausgehen, daß der Leser dieses Buchs die notwendige Ausstattung hat, die benötigt wird, um alles nachzuvollziehen, worüber hier berichtet wird.
Gleichwohl soll ein Überblick geboten werden, damit der Leser abschätzen kann, welche Weiterungen möglich sind.

8.2 Andere Programmiersprachen

Wenn man darauf verzichtet, mit dem Applesoft-BASIC-Interpreter zu arbeiten oder eine komfortablere BASIC-Version per Diskette zu laden, dann kann man auch auf andere Programmiersprachen zurückgreifen - vorausgesetzt, man hat die entsprechenden Übersetzungsprogramme (Compiler) auf Diskette und man verfügt über hinreichende Arbeitsspeicherkapazitäten (in der Regel mindestens 48 K).

Sind diese Voraussetzungen erfüllt, können eine Reihe verschiedener Programmiersprachen verwendet werden, die hier nicht alle aufgezählt werden sollen. Wir wollen vielmehr das Augenmerk des Lesers auf zwei spezielle Sprachen lenken, die in der Geschichte des Computereinsatzes bis in die heutige Zeit hinein eine besonders wichtige Rolle spielen. Es sind dies die Sprachen

FORTRAN und COBOL.

Beiden Sprachen ist gemeinsam, daß sie schon seit Jahrzehnten existieren, eine ungeheure Verbreitung (und damit Popularität) bei Großrechenanlagen und jetzt auch in zunehmendem Maße bei Kleinrechnern gefunden und ständig Erweiterungen, Ergänzungen, Verbesserungen und Modernisierungen erfahren haben.

FORTRAN steht als Abkürzung für

FORmula TRANslation ("Formel-Übersetzung")

und ist eine Sprache, die insbesondere für technisch-wissenschaftliche und vornehmlich mathematisch erfaßbare Probleme entwickelt wurde.

COBOL steht als Abkürzung für

Common Business Oriented Language
("Allgemeine betrieblich orientierte Sprache")

und dient vornehmlich zur Lösung von EDV-Problemen im betrieblich-kommerziellen Bereich.

Die Eigenschaften und Eigenheiten dieser beiden sehr charakteristischen Sprachen sollen im folgenden kurz skizziert werden - nicht etwa, um hier einen kompletten FORTRAN- oder COBOL-Kurs vorzuführen, sondern um dem Leser eine grobe Vorstellung davon zu vermitteln, inwieweit sich diese Sprachen von dem ihm inzwischen vielleicht lieb gewordenen oder zumindest gewohnten BASIC unterscheiden.

Beginnen wir zunächst mit der Programmiersprache FORTRAN. Ähnlich wie bei BASIC, gibt es auch hier verschiedene Dialekte. Wir beziehen uns im folgenden auf die neuere Version, nämlich auf FORTRAN 77.

Mit dieser FORTRAN-Version ist das sehr lang gebräuchliche FORTRAN IV ersetzt worden, das auf die modernen Möglichkeiten der dialogorientierten Programmierung nicht reagieren konnte.

Zwischen den Programmiersprachen BASIC und FORTRAN besteht eine relativ enge Verwandschaft.
Wie in BASIC gibt es auch hier bestimmte Anweisungsgruppen: Anweisungen

- zur Eingabe von Informationen,
- zur Ausgabe von Ergebnissen,
- zur Durchführung von Rechenprozeduren,
- zur Durchführung von Textverarbeitungsprozeduren,
- für Programmverzweigungen,
- für Programmsprünge
- für den Einbau von Unterprogrammen

etc.

Die Unterschiede zur Programmiersprache BASIC sind zunächst vor allem formaler Art:

FORTRAN-Statements werden nicht konsequent durchnumeriert, und wenn sie eine Nummer bekommen, dann dient diese nur als Ansprungadresse. Da FORTRAN-Statements immer mit sechs Leerzeichen eingeleitet werden, hat man in den vorderen sechs Anschlägen Platz genug für solche, in wenigen Fällen benötigten, Statement-Numnern; sie stehen dann in Anschlag 1-5 (Anschlag 6 ist für die Markierung eventueller Fortsetzungszeilen bei langen Statements vorgesehen).

Ohne in die Details zu gehen, sollen kurz die wichtigsten Statements und deren formaler Aufbau betrachtet werden, wobei wir - um dem Leser das Verständis für diese Statements zu erleichtern - die entsprechenden BASIC-Statements (so weit vorhanden) gegenüberstellen. Es versteht sich, daß wir bei dieser Aufzählung keine Vollständigkeit anstreben.

Die wichtigsten FORTRAN-Statements sind die folgenden:

1. Dateneingabe:

READ *, Variablenliste

Mit diesem Statement werden kleinere Datenbestände eingegeben.

READ (n,m) Variablenliste

Mit diesem Statement werden größere (und formatierte) Datensätze eingelesen. Dabei steht unter n die Gerätenummer des Eingabegeräts und unter m die Statementnummer der zugehörigen FORMAT-Angabe.

2. Formatierung:

m FORMAT (Spezifikationen)

Mit dieser Anweisung wird der Datensatz beschrieben, der (in der Regel bei mehrfacher Wiederholung mit unterschiedlichen Daten) eingelesen werden soll.
Nähere Erläuterungen dazu folgen nach den anderen Statements, die hier vorgestellt werden sollen.

Das FORMAT-Statement muß mit einer Statement-Nummer eingeleitet werden, die im zugehörigen READ-Statement (s.o.) auftaucht.

3. Datenausgabe:

PRINT *, Variablenliste

Statement zur Ausgabe von kleineren Ergebnismengen.

WRITE(n,m) Variablenliste

Ausgabe größerer Ergebnislisten in formatierter Weise (s.o.).

4. Arithmetische Anweisungen:

Variable = arithmetischer Ausdruck

Dies ist eine einfache Rechenanweisung; sie entspricht dem LET-Statement der Sprache BASIC.

5. Bedingter Sprung:

IF (logischer Ausdruck) Folgeanweisung

Dieses Statement entspricht dem IF...THEN-Statement aus BASIC, wobei aber andere logische Operatoren verwendet werden:

.EQ.	entspricht	=
.NE.	"	<>
.LE.	"	<=
.GE.	"	>=
.LT.	"	<
.GT.	"	>

6. Unbedinger Sprung:

GO TO m

Auch diese Anweisung unterscheidet sich nicht vom GO TO-Statement aus BASIC.

7. Arithmetische Verzweigung:

IF (arithm. Ausdruck) n_1, n_2, n_3

Diese sog. arithmetische IF-Anweisung verzweigt zum Satz Nr. n_1, wenn der arithmetische Ausdruck negativ ist, zum Satz n_2, wenn er null ist und zum Satz n_3, wenn er positiv ist.

8. Schleife:

DO n, Variable = Anfang, Ende

Mit diesem Statement wird eine Schleife eingeleitet (entsprechend FOR...TO... in BASIC).

n CONTINUE

Statement zur Beendigung einer Schleife (entsprechend dem NEXT-Statement in BASIC);
die Statementnummer n muß der Nummer n im DO-Statement entsprechen.

9. Dimensionierung:

DIMENSION Variablenname (Wert)

Diese Anweisung entspricht dem DIM-Statement aus der Sprache BASIC.

1o. Datenbereitstellung:

DATA Variablenliste/Werte/

Dieses Statement erlaubt Datenzuweisungen, ohne daß per READ gelesen werden müßte; insoweit handelt es sich hier, würde man mit BASIC vergleichen, um eine Koppelung von DATA- und READ-Statement.

CHARACTER Variable = Ausdruck

Mit dieser Anweisung sind in FORTRAN Textzuweisungen möglich (entsprechend der Anweisung LET Var.name$ = String in BASIC).

11. Unterprogramme:

SUBROUTINE Name (Argumente)

Mit diesem Statement werden Unterprogramme eingeleitet (die, ebenso wie das Hauptprogramm, in FORTRAN mit END beschlossen werden).

Der Aufruf des Unterprogramms aus dem Hauptprogramm erfolgt mit:

CALL Name (Argumente)

(Dies entspricht dem GOSUB-Statement aus BASIC).

12. Programmbeendigung:

Das FORTRAN-Programm wird (wie in BASIC) mit dem Statement

```
END
```

beendet.

13. Kommentare:

Diese - noch sehr unvollständige Aufzählung - soll mit einer FORTRAN-Möglichkeit abgeschlossen werden, die dem REM-Statement aus BASIC entspricht:

Alle Texte, die in Anschlag 1 mit einem C eingeleitet werden, dienen als Kommentare und werden deshalb genauso behandelt wie REM in der Sprache BASIC.

Es fällt bei dieser Aufzählung besonders ins Auge, daß in FORTRAN im Unterschied zu BASIC ein sog. FORMAT-Statement benutzt werden kann.

Dieses FORMAT-Statement dient dazu, einen Datensatz zum Zwecke des Einlesens und/oder zum Zwecke der Ausgabe zu formatieren. Betrachten wir dazu zur Verdeutlichung der Vorgehensweise das folgende Beispiel zweier Datensätze, die willkürlich aus einem Datenbestand herausgegriffen wurden:

Nr.	Name	Alter	Größe	Gewicht
17	MUELLER	42	1.85	84.5
18	ROLF	38	1.7Ø	75

In BASIC würde man diese Datensätze folgendermaßen dem Rechner übergeben können:

```
 1Ø DATA 17,MUELLER,42,1.85,84.5
 2Ø DATA 18,ROLF,38,1.7,75
  :
499 Z=2
5ØØ FOR I=1TOZ
51Ø READ NR(I),NØ(I),A(I),GR(I),GW(I)
52Ø NEXT I
```

In FORTRAN hingegen gibt man in der Regel den Datenbestand gesondert vor, und zwar in der Praxis meist im "fixed format", d.h. mit konstanten Feldlängen, also z.B. folgendermaßen:

```
17 MUELLER        421.8584.5
18 ROLF           381.7 75
```

Dies bedingt, daß man im Programm dem Rechner die Feldeinteilung und auch den Typ der jeweiligen Feldvariablen (numerisch reell, numerisch ganzzahlig oder alphanumerisch) mitteilt.

Genau dies leistet das FORMAT-Statement.

Per Voreinstellung gilt in FORTRAN folgende Regel:

Alle Variablennamen, die mit den Buchstaben I bis N beginnen, werden als ganzzahlige Variablen betrachtet (I-N = IN = Integer = ganzzahlig), alle anderen als reelle Variablen (von dieser Voreinstellung kann mit geeigneten Typvereinbarungs-Statements bei Bedarf abgewichen werden; dies soll hier aber nicht besprochen werden).

Ganzzahlige Variablen werden mit dem sog. I-Format, reelle Variablen mit dem F-Format, String-Variablen mit dem A-Format behandelt (es gibt eine Vielzahl weiterer Möglichkeiten - etwa zum Übergehen von Feldern, zum Zeilenwechsel, zur Behandlung doppelt genauer reeller Größen und andere - auch darüber soll hier nicht gesprochen werden).

Somit könnten die obigen Datensätze mit einem einzigen (wegen des "fixed format"), dem folgenden FORMAT-Statement beschrieben werden:

```
FORMAT  (I2, A15, I2, F4.2, F4.1)
```

Dabei bedeutet:

I2 : Das erste Feld umfaßt 2 Anschläge und es handelt sich um eine ganzzahlige Größe (deshalb I für "Integer");

A15 : Das zweite Feld ist ein alphanumerisches Feld, für das 15 Anschläge generell reserviert wurden;

I2 : Im dritten Feld steht wieder eine zweistellige ganzzahlige Größe;

F4.2 : Im folgenden Feld steht eine vierstellige reelle Größe (incl. Dezimalpunkt), wobei die beiden hinteren Stellen als Dezimalstellen zu interpretieren sind;

F4.1 : Entsprechend wie oben, allerdings wird hier nur eine Dezimalstelle mitgeführt, dafür aber zwei Stellen vor dem Dezimalpunkt.

Das Einlesen geschieht jetzt z.B. folgendermaßen:

```
      7
C     EINLESEN
      Z=2
      DO 1 I=1,Z
      READ (5,99)NR(I), N(I), A(I), GR(I), GW(I)
  99  FORMAT (I2, A15, I2, F4.2, F4.1)
 1    CONTINUE
C     WEITERE VERARBEITUNG
      .
      .
      .
```

Es bräuchte in diesem Programmsegment nur Z geändert zu werden, wenn man einen größeren, also einen "echten" Datenbestand einlesen wollte. Anzumerken ist, daß wir dann dimensionieren müssen, wie das ja auch in BASIC der Fall ist.

Betrachten wir abschließend ein einfaches FORTRAN-Programmbeispiel, dem wir das entsprechende BASIC-Programm gleich folgen lassen, so daß der Leser sofort nachvollziehen kann, was im einzelnen geschieht.

```
C     PROGRAMM ZUR BESTIMMUNG DER SUMME
C     ALLER GANZEN ZAHLEN VON 1 BIS 1ØØ
C
C
      DO 1Ø I=1,1ØØ
      S=S+I
  1Ø  CONTINUE
      WRITE (6,1ØØ)5X, "SUMME =",S
 1ØØ  FORMAT I6
      END
```

Das entsprechende BASIC-Programm lautet:

```
1Ø  REM PROGRAMM ZUR BESTIMMUNG DER SUMME
2Ø  REM ALLER GANZEN ZAHLEN VON 1 BIS 1ØØ
3Ø  REM
4Ø  FOR I=1 TO 1ØØ
5Ø  S=S+I
6Ø  NEXT I
7Ø  PRINT TAB(5)"SUMME =";S
8Ø  END
```

Auch die Programmiersprache COBOL, die wir uns ebenfalls in diesem Kapitel kurz anschauen wollen, weist gewisse Ähnlichkeiten zur Sprache BASIC auf; allerdings sind hier schon die formalen Unterschiede stärker ausgeprägt als im Vergleich zwischen FORTRAN und BASIC.

Deshalb wollen wir hier die wichtigsten Statements nicht, wie bei der kurzen Besprechung der Sprache FORTRAN, einfach aufzählen, sondern sie anhand der Entwicklung eines Programmbeispiels kennenlernen.

Der erste und wichtigste Unterschied zwischen BASIC und COBOL kommt dadurch zum Ausdruck, daß jedes COBOL-Programm grundsätzlich in vier Abschnitte aufgeteilt wird, die man Divisionen nennt.
Es handelt sich um folgende Abteilungen:

1. IDENTIFICATION DIVISION
2. ENVIRONMENT DIVISION
3. DATA DIVISION
4. PROCEDURE DIVISION

Mit diesen Überschriften müssen die vier Divisionen eingeleitet werden, die ihrerseits in Paragraphen unterteilt werden können. Dabei dienen die Paragraphennamen als Ansprungadressen oder anders ausgedrückt: Programmteile, die während der Programmabarbeitung angesprungen werden sollen, müssen ihrerseits als Paragraphen gekennzeichnet sein, um über eine Ansprungadresse verfügen zu können.

Weiterhin ist unter formalen Gesichtspunkten zu beachten, daß COBOL-Statements in Anschlag 8 oder 12 beginnen (sog. A-Rand und B-Rand; auf den Unterschied kommen wir noch zu sprechen), und daß sie mit einem Punkt zu beschließen sind.

Was geschieht nun in den einzelnen Divisionen?

Zunächst ist festzuhalten, daß es in den einzelnen Divisionen Anweisungen gibt, die obligatorisch sind, die also vom Programmierer angegeben werden müssen, während andere Anweisungen nur bei Bedarf in das COBOL-Programm aufgenommen werden.

Wir werden uns im folgenden vornehmlich mit den ersteren beschäftigen.

In der IDENTIFICATION DIVISION erhält das Programm einen Namen und es können weitere Anweisungen aufgenommen werden, die das Programm näher erläutern. Die Namensvergabe erfolgt mit dem Statement

PROGRAM-ID. Name (beliebiger Kommentar)

welches obligatorisch ist. Alle weiteren Angaben sind wahlfrei und sollen deshalb hier nicht betrachtet werden.

Die Angaben in der IDENTIFICATION DIVISION beginnen am sog. A-Rand, also ab Anschlag 8 und werden durch Punkt und Leerzeichen abgeschlossen (eventuelle Fortsetzungszeilen beginnen am B-Rand, also ab Anschlag 12).

Der Anfang eines COBOL-Programms könnte also folgendermaßen aussehen:

```
8
IDENTIFICATION DIVISION.
PROGRAM-ID. PROGRAMM NR.1.
```

In der ENVIRONMENT DIVISION wird die jeweils benutzte Gerätekonfiguration erläutert, d.h. es werden Input-, Output- und ggf. Speichergeräte mit ihren logischen Gerätenummern genannt.

Im einzelnen wird in dieser Division festgehalten, auf welchem Computer das Programm compiliert und auf welchem es ausgeführt wird. Darüber hinaus wird die Beziehung zwischen den verwendeten Dateien (häufig eine Input- und eine Outputdatei) und den Geräten der Peripherie, die für diese Dateien zuständig sein sollen, hergestellt.

Da jeder Computerhersteller die Bezeichnungen und Gerätenummern der peripheren Einheiten unterschiedlich festlegen kann, hängt der Inhalt der ENVIRONMENT DIVISION stark vom jeweils benutzten Rechner ab.

Diese Division wird in zwei <u>Sektionen</u> unterteilt:

1. CONFIGURATION SECTION
   (sie beschreibt die benutzte Rechenanlage)
2. INPUT-OUTPUT SECTION
   (sie beschreibt die Datei-Geräte-Zuordnung).

Unser oben begonnenes Programm könnte nun wie folgt fortgesetzt werden:

```
        8
(1)     IDENTIFICATION DIVISION.
(2)     PROGRAM-ID.  PROGRAMM NR.1.
(3)     INVIRONMENT DIVISION.
(4)     CONFIGURATION SECTION.
(5)     SOURCE COMPUTER.  APPLE.
(6)     OBJECT COMPUTER.  APPLE.
(7)     INPUT-OUTPUT SECTION.
(8)     FILE-CONTROL.
(9)         SELECT EINGABE ASSIGN TO gerät.
(1Ø)        SELECT AUSGABE ASSIGN TO Gerät.
```

Um den Programmausschnitt leichter erläutern zu können, haben wir am linken Rand numeriert:

DIESE NUMERIERUNG IST NICHT BESTANDTEIL DES COBOL-PROGRAMMS!

Die Sätze (1) und (2) sind uns schon bekannt.
Mit Satz (3) wird die zweite Division eingeleitet, mit Satz (4) deren erste Sektion.

In Satz (5) wird der Übersetzungs-Computer, in Satz (6) der Computer, auf dem gerechnet werden soll, genannt. Häufig dürften beide identisch sein.
Von der Reihenfolge der Anweisungen, wie sie sich oben darstellt, darf nicht abgewichen werden.

Mit Satz (7) wird die zweite Sektion der zweiten Division gekennzeichnet. Ihr erstes obligatorisches Statement ist das FILE-CONTROL-Statement, das die Datei-Zuordnungen einleitet.

In Satz (9), der wie auch Satz (1Ø) quasi Unterbestandteil der FILE-CONTROL-Anweisung ist, und deshalb ab Anschlag 12 beginnt, werden die Geräte zugeordnet.

Allgemeine Schreibweise:

SELECT Name der Datei ASSIGN TO Gerätenummer

Welche Geräteangabe von Fall zu Fall einzugeben ist, muß den jeweiligen Benutzerhandbüchern entnommen werden.

In der DATA DIVISION wird der Aufbau eines Datensatzes detailliert vorgestellt (insoweit etwa vergleichbar mit dem FORTRAN-Statement FORMAT).

Dabei spielt das sog. Stufenkonzept eine wesentliche Rolle: Der jeweilige Datensatz wird in Datengruppen unterteilt, diese wiederum in Untergruppen etc.

Das folgende Beispiel illustriert die Vorgehensweise.

Gegeben sei folgender Datensatz aus einer Kundendatei:

Nr.	Name		Adresse				...
	Vorname	Nachname	Post-leitzahl	Ort	Wohnung		
					Straße	Nr.	

Man erkennt hier eine maximal dreistufige Untergliederung, die aber nicht in jeder Datengruppe "durchgehalten" werden muß.

Ähnlich wie im FORMAT-Statement aus FORTRAN muß nun dem Rechner die Feldaufteilung mitgeteilt werden. Dies geschieht mit der sog. PICTURE-Klausel (PICTURE = Bild), wie das folgende Beispiel, von dem obigen Datensatz ausgehend, zeigt:

Stufenkonzept-Beispiel:

```
Ø1  KUNDENSATZ.
    Ø2  NR PICTURE IS 9(4).
    Ø2  NAME.
        Ø3  VORN PICTURE IS X(1Ø).
        Ø3  NACHN PICTURE IS X(15).
    Ø2  ADRESSE.
        Ø3  PLZ PICTURE IS 9(4).
        Ø3  ORT PICTURE IS X(2Ø).
        Ø3  WOHNUNG.
            Ø4  STR PICTURE IS X(15).
            Ø4  HNR PICTURE IS 9(3).
```

Wir erkennen eine stufenweise Untergliederung gemäß des schematischen Aufbaus eines Datensatzes, wie er weiter oben vorgestellt wurde.

Diejenigen Felder, die weiter unterteilt werden (gleichgültig auf welcher Stufe sie sich befinden), erhalten keine PICTURE-Klausel (z.B. Ø2 NAME.); diejenigen hingegen, die nicht mehr aufgeteilt werden (z.B. Ø3 PLZ), erhalten die PICTURE-Klausel, die dem Rechner sagt, wie das jeweilige Feld aussieht (diese Felder haben wir im Schema weiter oben doppelt unterstrichen; diese Unterstreichungen folgen dem unteren Rand dieses Schemas).

Bei den PICTURE-Klauseln wird die Ziffer 9 zur Kennzeichnung numerischer Felder, der Buchstabe X zur Kennzeichnung alphanumerischer Felder verwendet (es gibt viele weitere Kennungsmöglichkeiten, z.B. zur Kennzeichnung von Dezimalpunkt und Dezimalstellen u.ä.).

PICTURE IS 9(4) bedeutet also z.B., daß ein vierstelliges numerisches Feld betrachtet wird (statt 9(4) dürfte auch 9999 geschrieben werden).

Nur am Rande sei hier schon darauf hingewiesen, daß auf dieses Stufenkonzept völlig verzichtet werden kann, wenn nur kleine Datenmengen ein- oder ausgegeben werden sollen (z.B. dialogorientiert). Darauf kommen wir weiter unten zu sprechen.

Es versteht sich, daß sowohl für den einzugebenden Datenbestand eine derartige Strukturierung vorgenommen werden muß, wie auch für den Datenbestand, der als Ergebnis des Datenverarbeitungsprozesses ausgegeben werden soll. Werden mehr als zwei Dateien (Input- und Outputdatei) in einem Datenverarbeitungsprozeß verwendet, benötigt oder angelegt, dann braucht man natürlich noch weitere derartige Strukturierungen.

Diese Strukturierungen müssen - wenn sie überhaupt erforderlich sind - in der ersten Sektion der DATA DIVISION erfolgen, die die Überschrift FILE SECTION trägt. Ihr erstes Statement ist das sog. FD-Statement (FD = File-Declaration). Es lautet allgemein:

```
FD  Dateiname  Dateibeschreibung
```

In der Datenbeschreibung wird z.B. mitgeteilt, wieviele Datensätze die Datei enthält, wie lang ein Datensatz ist, ob die Datensätze sequentiell oder nicht gespeichert sind, ob und ggf. wie die Datei geschützt ist u.ä. Auf Einzelheiten dazu soll hier nicht eingegangen werden.

Die zweite Sektion der DATA DIVISION ist die WORKING-STORAGE SECTION, in der die Speicherplätze für all diejenigen Daten reserviert und beschrieben werden, die nicht über Ein- oder Ausgabedateien verwaltet werden.

Es handelt sich dabei um Hilfsfelder, die im Laufe des Datenverarbeitungsprozesses nach der Eingabe und vor der Ausgabe benötigt werden, bzw. um Ein- und/oder Ausgabefelder bei kleinen Datenmengen, die nicht über Dateien organisiert wurden.

In der WORKING-STORAGE SECTION wird ebenfalls mit dem Stufenkonzept gearbeitet, allerdings entfällt natürlich das FD-Statement.
Hier wird es relativ häufig geschehen, daß man auch für sich allein stehende Werte betrachtet, die also nicht als Teil eines Stufenkonzepts einzuordnen sind. Für solche Felder ist die Stufennummer 77 reserviert (sie kann für den gleichen Zweck auch in der FILE SECTION verwendet werden).

<u>Beispiel:</u>

```
8
WORKING-STORAGE SECTION
77  ZAHL1  PICTURE IS  9999.
77  ZAHL2  PICTURE IS  9999.
Ø1  SATZ.
    Ø2 MULT PICTURE IS 9(8).
    Ø2 DIV  PICTURE IS 999V99.
```

(Das V in der letzten PICTURE-Klausel steht für die Kennzeichnung der Lage eines Dezimalpunkts.)

Wenden wir uns nun der letzten Division, der PROCEDURE DIVISION zu: Hier finden sich die Anweisungen zur Verarbeitung der eingegebenen Daten. Die wichtigsten dieser Anweisungen sind die folgenden:

1. Dateieröffnung:

Mit der OPEN-Anweisung müssen Ein- und Ausgabedateien, sofern sie benutzt werden sollen, eröffnet werden.

Beispiel:

```
OPEN INPUT EINGABE.
OPEN OUTPUT AUSGABE.
```

Allgemein:

```
OPEN [INPUT  ] Dateiname.
     [OUTPUT ]
```

2. Schließen von Dateien:

Jede geöffnete Datei muß vor Programmende wieder geschlossen werden:

Allgemein: CLOSE Dateiname.

3. Lesen von Eingabedateien:

Mit der READ-Anweisung werden die Datensätze der Eingabedatei gelesen.

Allgemein: READ Dateiname RECORD AT END Anweisung.

4. Ausgabe von Datensätzen:

Mit der WRITE-Anweisung werden Ergebnisse ausgegeben.

Allgemein: WRITE Datensatzname.

Im Gegensatz zur READ-Anweisung taucht hier also nicht der Dateiname der Ausgabedatei auf, sondern der Datensatzname (er entspricht derjenigen Feldbezeichnung, die im Stufenkonzept der Ausgabedateibeschreibung in der DATA DIVISION die Stufennummer Ø1 trägt).

Der Sinn dieses Unterschieds liegt darin, daß per READ die gesamte Eingabedatei gelesen wird (und gelesen werden soll), während per WRITE nur jeweils Datensatz für Datensatz (also immer nur jeweils einer) ausgegeben wird.

5. Die MOVE-Anweisung:

Allgemein: MOVE Sendefeld TO Empfangsfeld

Diese Anweisung entspricht dem LET-Statement aus BASIC.

Es gibt auch eine einfachere Möglichkeit der Wertzuweisung, die sich bei kleinen Datenmengen empfiehlt und dann schon in der WORKING-STORAGE SECTION der DATA DIVISION erfolgt.

Beispiel:

```
WORKING-STORAGE SECTION
77  ZAHL1  PICTURE IS 9999 VALUE IS 1234.
77  NAME1  PICTURE IS X(4) VALUE IS 'VOSS'.
```

6. Arithmetische Ausdrücke:

Die Grundrechenarten werden in COBOL nach dem Muster der folgenden Beispiele durchgeführt:

Addieren: ADD a TO b.
Rechnung: b = a+b

Subtrahieren: SUBTRACT a FROM b.
Rechnung: b = b-a

Multiplizieren: MULTIPLY a BY b.
Rechnung: b = a*b.

Dividieren: DIVIDE a INTO b.

Rechnung: $b = \frac{b}{a}$

Eine alternative Formulierungsmöglichkeit, die für alle vier Rechenarten entsprechend gilt, soll nur am Beispiel des Addierens demonstriert werden:

ADD a b GIVING c
Rechnung: c = a+b

Für kompliziertere Berechnungen empfiehlt sich die COMPUTE-Anweisung:

COMPUTE a = arithmetischer Ausdruck

7. Bedingter Sprung:

Die allgemeine Form des sog. abhängiggen GO TO lautet:

GO TO Liste DEPENDING ON Variable

Hat "Variable" den Wert 1, wird der Paragraph mit dem ersten Namen aus "Liste" angesprungen; hat "Variable" den Wert 2, erfolgt ein Sprung zum zweiten Namen aus "Liste" usw.

Entsprechend gibt es auch eine IF-Anweisung, die folgende allgemeine Form aufweist:

IF Bedingung [Anweisung / NEXT SENTENCE] (ELSE...)

Diese Anweisung funktioniert im Prinzip wie die IF...THEN(...ELSE...)-Anweisung aus BASIC.

Als Vergleichsoperatoren können verwendet werden:

= > <

8. Beendigung des Programms:

Ein COBOL-Programm wird mit der Anweisung

```
STOP RUN.
```

beendet.

Dies sind die wichtigsten (aber keineswegs alle) COBOL-Anweisungen.
Sie sollen noch um die beiden Anweisungen ergänzt werden, mit denen kleine Datenmengen ein- bzw. ausgegeben werden können. In der DATA DIVISION muß dann lediglich in der WORKING-STORAGE SECTION Platz geschaffen werden und die etwas aufwendige Stufenkonzeptbildung wird entbehrlich.

Eingabe:	ACCEPT Variablenname.
Ausgabe:	DISPLAY ('Text') Variablenname.

Betrachten wir auch hier abschließend ein einfaches Beispiel, dem wir wieder das entsprechende BASIC-Programm gegenüberstellen:

```
8
INDIFICATION DIVISION
PROGRAM-ID.QUADRIERUNG.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. APPLE.
OBJECT-COMPUTER. APPLE.

DATA DIVISION.

WORKING-STORAGE SECTION.
77  EIN  PICTURE IS 9(4).
77  AUS  PICTURE IS 9(8).

PROCEDURE DIVISION.
ANFANG.
    ACCEPT EIN.
    COMPUTE AUS= EIN+EIN.
    DISPLAY 'QUADRAT=',AUS.
ENDE.
    STOP RUN.
```

In diesem Programm wird eine einzelne einzugebende Zahl quadriert und das Rechenergebnis direkt ausgegeben.

In der Programmiersprache BASIC kann diese Aufgabe durch das folgende einfache Programm erledigt werden.

```
1Ø  INPUT "BITTE EINE ZAHL :";E
2Ø  A = E*E
3Ø  PRINT "QUADRAT =";A
4Ø  END
```

Wir sehen, daß dieses BASIC-Programm im Vergleich viel einfacher ausfällt. Die Vorteile der COBOL-Programmierung erweisen sich in der Tat erst dann, wenn sehr große Datenmengen zu verarbeiten sind und wenn man die zusätzlichen und speziellen COBOL-Möglichkeiten kennengelernt hat.

8.3 Software

Für BASIC-Rechner, wie z.B. den APPLE II, steht eine Reihe von qualifizierten Software-Angeboten zur Verfügung. Dieses Angebot ist so umfangreich, daß es völlig ausgeschlossen ist, hier einen kompletten Überblick bieten zu wollen.

Einige Stichworte sollen jedoch eine erste Vorstellung von den Möglichkeiten vermitteln, die sich bieten.

Software-Angebote können einmal danach unterteilt werden, in welchen Praxisbereichen sie sinnvoll eingesetzt werden, zum anderen danach, welche der prinzipiellen Computeraktivitäten mit diesen Programmangeboten unterstützt werden.

Bezüglich der Praxisbereiche bietet sich die folgende Einteilung an (wobei wiederum kein Anspruch auf Vollständigkeit dieser Aufzählung erhoben wird):

1. Programme zur Lösung mathematischer Probleme;
2. Programme im betrieblichen Einsatz (z.B. Buchhaltung, Lagerverwaltung, Kundendateien, Kalkulation u.ä.);
3. Programme für statistische Auswertungen;
4. Programme zur Unterstützung von Büro- und Verwaltungsarbeiten;

5. Programme zur Erstellung von Planungen, Entwürfen, Skizzen, Zeichnungen und dergl.;

6. Programme zur Steuerung von Automaten und Maschinen;

7. Nicht zu vergessen, aber vielleicht nicht so relevant, die Spielprogramme.

Betrachtet man die Computeraktivitäten, die per Software unterstützt werden, so bietet sich die folgende Einteilung an:

1. Rechenprogramme

2. Textverarbeitungsprogramme

3. Graphikprogramme.

Dazu einige besonders typische Beispiele:

Als "Rechenprogramme" können beispielsweise alle Kalkulationsprogramme bezeichnet werden, die für den APPLE II zur Verfügung stehen und die insbesondere im betrieblichen Bereich eine große Bedeutung erlangt haben. Wesentliche Vertreter dieser Programmgruppe sind etwa die Programme

VisiCalc (vorgesehen für APPLE III)
und MULTIPLAN

Speziell MULTIPLAN ist ein Kalkulationsprogramm für den APPLE IIe. Es eignet sich für Tabellenauswertungen, Budgetplanungen, Trendberechnungen und dergleichen (auf Einzelheiten dieser und der folgenden Programme soll nicht eingegangen werden).

Wenn wir den APPLE II mittels einer Zusatzkarte auf ein anderes Betriebssystem umstellen, dann kann man auch weitere, teilweise sehr leistungsfähige Kalkulationsprogramme einsetzen.
Es versteht sich, daß auch branchenspezifische Programme zur Verfügung stehen (von Programmen für Architekten und Ärzten bis hin zum Verlags- und Versicherungswesen). Auch darüber soll aber hier nicht gesprochen werden.

Zu den Rechenprogrammen wird man auch alle diejenigen Programme zählen, die für spezielle mathematische Aufgaben entwickelt wurden. Beispiele dafür sind Programme für Vektor- und Matrizenrechnungen, Programme für Differential- und Integralrechnung und dergl.

Ein spezielles Gebiet in diesem Zusammenhang ist die mathematische Statistik. Hier stehen Programme für unterschiedliche Zwecke zur Verfügung.

1. Dialogorientierte Dateneingabe, Datenkontrolle und deskriptive statistische Analyse (z.B. System DATAN II von R. Schnell);

2. Inferenzstatistische Datenanalyse durch Einsatz der bekannten statistischen Testverfahren (z.B. System REPTEST von R.Prust);

3. Multivariate statistische Analyseverfahren (z.B. System TAXONOM von Weber)

u.ä.

Auch für den Bereich der Textverarbeitung stehen Programme zur Verfügung, die entweder speziell auf das Betriebssystem des APPLE II abgestellt sind oder die nach Implementierung eines anderen Betriebssystems benutzt werden können.

Zur ersten Gruppe gehören die Programme APPLE WRITER und SCRIPT; zur zweiten Gruppe gehört das berühmte und weitverbreitete System WORDSTAR.

Mit Textverarbeitungssystemen kann man Briefe, Massenschreiben, Rechnungen und dergl. abfassen, aber auch Manuskripte erstellen, wissenschaftliche Texte u.ä. Ohne Probleme sind Schriftartenwechsel möglich, nachträgliche Fehlerkorrekturen, Hinzufügen oder Streichen von Texten, Anfügung von Fußnoten, Einfügung von Sonderzeichen, Unterstreichungen usw.
Einzelheiten sind den jeweils mitgelieferten Handbüchern zu entnehmen.

Eine spezielle Gruppe von Programmen dient der Dateiverwaltung. Man versteht darunter alle diejenigen Programme, mit denen größere Datenmengen geschickt eingegeben und gespeichert werden und mit denen die so entstehenden Dateien dann optimal verwaltet werden können.

Dies betrifft beispielsweise die Erweiterung oder Verkürzung der Datei, ihre Veränderung und vor allem das Aufsuchen einzelner Datensätze nach vorgegebenen und miteinanderkombinierbaren Suchkriterien und Such-Stichworten.

Typische Vertreter derartiger Programme für den APPLE II sind etwa die Programme QuickFile und ProFit II.

Die letzte genannte Gruppe betrifft die Graphikprogramme. Wie wir in einem früheren Kapitel gesehen haben, ist der APPLE II graphikfähig und es stehen eine Reihe von Anweisungen zur Verfügung, mit denen graphische Ausgaben erzeugt werden.

Zusätzlich gibt es Hilfsprogramme etwa zur Erzeugung betrieblich nutzbarer Graphiken (Zeitreihen, Kurven- und Balkendiagramme, Kreisdiagramme, Beschriftungen, Farbgraphiken, Schraffuren usw.) oder zur Erzeugung architektonischer oder konstruktionstechnischer Zeichnungen (über künstlerische Graphiken oder die Bedeutung der Graphikprogrammierung im breiten Bereich der Computerspiele soll hier nicht gesprochen werden).

Zur ersten Gruppe gehören Programme, wie z.B. Business Graphics oder CHARTMASTER (beide für APPLE IIe oder APPLE III).
Zur zweiten Gruppe gehören die neu auf den Markt kommenden Programme, die für die CAD-Programmierung (CAD = Computer Aided Design = Computergestütztes Entwickeln und Konstruieren) bereitgestellt werden.
Auf Einzelheiten derartiger Programme soll hier nicht mehr eingegangen werden.

A N H Ä N G E

Anhang I: Fehlermeldungen (Auswahl)

BAD SUBSCRIPT ERROR

Eine Variable enthält weniger oder mehr Indizes als in der DIM-Anweisung vorgesehen ist oder ein Indexwert ist größer als er aufgrund der DIM-Anweisung sein darf.

CAN'T CONTINUE ERROR

Das Kommando CONT kann nach einer Programmunterbrechung nicht ausgeführt werden (etwa, weil in der Zwischenzeit eine Programmänderung vorgenommen wurde).

DISK FULL

Die Diskette ist voll belegt. Weitere Informationen können nicht gespeichert werden.

DIVISION BY ZERO ERROR

Es wurde versucht, durch Null zu dividieren.

END OF DATA

Es wurde versucht, von einer Diskette Daten zu lesen, die dort nicht gespeichert sind.

Anhang I

FILE LOCKED

Es wurde versucht, eine schreibgeschützte Diskettendatei zu verändern.

FILE NOT FOUND

Es wurde versucht, auf eine nicht existierende Datei zurückzugreifen.

FILE TYPE MISMATCH

Es wurde eine Datei mit einer Anweisung angesprochen, die für diese Datei nicht zulässig ist.

ILLEGAL QUANTITY ERROR

Ein Zahlenwert liegt außerhalb des zulässigen Bereichs.

I/O ERROR

Eine Diskette wurde angesprochen, aber nicht erreicht (Beispiel: Nichtinitialisierte Diskette).

Anhang I

NEXT WITHOUT FOR ERROR

Es wurde eine NEXT-Anweisung erreicht ohne dazugehöriges FOR...TO...Statement.

OUT OF DATA ERROR

Es wurde versucht, mehr Daten zu lesen als vorhanden sind.

OUT OF MEMORY ERROR

Die Speicherkapazitäten reichen für ein bestimmtes Programm nicht aus.

OVERFLOW ERROR

Eine Zahl ist zu groß (größer als |1.7E+38|).

REDIM'D ARRAY ERROR

Es wurde versucht, eine bereits dimensionierte Variable erneut zu dimensionieren.

RETURN WITHOUT GOSUB ERROR

Es wurde eine RETURN-Anweisung ohne dazugehörige GOSUB-Anweisung gefunden.

Anhang I

SYNTAX ERROR

Formaler Fehler in einer Anweisung oder Fehler, für den es keine spezielle Fehlermeldung gibt.

TYPE MISMATCH ERROR

Numerische und String-Angaben wurden unzulässigerweise in einer Anweisung gemischt.

UNDEFINED FUNCTION ERROR

Es wurde versucht, eine nicht definierte Funktion aufzurufen.

UNDEFINED STATEMENT ERROR

Eine Anweisung verzweigt zu einer nicht existierenden Satznummer.

Anhang II: Wichtige Unterprogrammadressen

CALL -	1Ø52	Piepston erzeugen
CALL -	1ØØ8	Cursor eine Spalte nach links verschieben
CALL -	998	Cursor eine Zeile nach oben verschieben
CALL -	958	Bildschirm zwischen Cursor und unterem Rand löschen
CALL -	922	Cursor eine Zeile nach unten verschieben
CALL -	868	Bildschirmzeile rechts vom Cursor löschen
CALL -	336	Aufrufen des BASIC-Interpreters
CALL -	211	ERR ausgeben und Piepston erzeugen
CALL -	151	Monitor aufrufen

Anhang III: Speicheradressen (Auswahl

Adressen		Funktion
Dezimal	Hexadezimal	
32	$2Ø	Linker Bildschirmrand; zulässige Werte: Ø...39
33	$21	Bildschirmbreite; zulässige Werte: Ø...(4Ø-PEEK(32))
34	$22	Oberer Bildschirmrand; zulässige Werte: Ø...23
35	$23	Unterer Bildschirmrand; zulässige Werte: Ø...23, wenn die folgende Bedingung erfüllt ist: PEEK(34) < PEEK(35)
36	$24	Horizontale Cursorposition; zulässige Werte: Ø...PEEK(33)
37	$25	Vertikale Cursorposition; zulässige Werte: Ø...23
5Ø	$32	Ausgabemodus für den Bildschirm; mögliche Werte: 63 - INVERSE, 127 - FLASH, 255 - NORMAL
1Ø24 bis 2Ø47	$4ØØ bis $7FF	Bildschirmspeicherbereich für die Textseite 1; zulässige Werte: Alle ASCII-Bildschirmcodezahlen

Anhang III

Adressen		Funktion
Dezimal	Hexadezimal	
-16368 (49168)	$C01Ø	Flag für erfolgte Tastatureingabe
-16384 (49152)	$CØØØ	Enthält den Code des letzten eingegebenen Zeichens
1Ø3/1Ø4	$67/68	Programmanfang
1Ø5/1Ø6	$69/6A	LOMEM - enthält die kleinste von BASIC ansprechbare Adresse
115/116	$73/74	HIMEM - enthält die größte von BASIC ansprechbare Adresse
175/176	$AF/BØ	Erste Adresse hinter dem Programmende; in der Regel also Beginn der Variablenwerte
1Ø9/11Ø	$6D/6E	Ende des benutzten numerischen Speichers
218/219	$DA/DB	Zeilennummer, in der der letzte Fehler aufgetreten ist
222	$DE	Codeziffer des letzten Fehlers
216	$D8	Errorflag für ONERR...GOTO-Befehle

Anhang IV: Speichereinteilung des APPLE II

Adressen dezimal	Adressen hexadezimal	Art d.Speichers	Verwendung
0-255	0-FF	S/L	Systemprogramme
256-511	100-1FF	S/L	Stapel
512-767	200-2FF	S/L	Eingabepuffer für Tastatur
768-1023	300-3FF	S/L	Zeiger des Monitors
1024-2047	400-7FF	S/L	Ausgabepuffer 1 für Textdaten und graphische Daten niedriger Auflösung
2048-3071	800-BFF	S/L	Ausgabepuffer 2 für Textdaten und graphische Daten niedriger Auflösung
3072-8191	C00-1FFF	S/L	frei
8192-16383	2000-3FFF	S/L	Ausgabepuffer 1 für graphische Daten hoher Auflösung
16384-24575	4000-5FFF	S/L	Ausgabepuffer 2 für graphische Daten hoher Auflösung
24576-49151	6000-BFFF	S/L	frei
49152-53247	C000-CFFF		Ein/Ausgabeadressen
53248-65535	D000-FFFF	Festwert	Integer-BASIC-Interpreter, Applesoft-Interpreter, Monitor etc.

Stichwortverzeichnis

** A **

A-RAND 366
ABS 13
ACCEPT 378
ADD 376
ADRESSE 130
AENDERUNG 298
ALGOL 113
ANWEISUNG 8
ANWEISUNG, ARITHMETISCHE 357
APA 263, 265
APPEND 64, 72
APPLE-DOC 84, 263, 288
APPLE-IIC 4
APPLE-WRITER 383
ARITHMETISCHE ANWEISUNG 357
ARITHMETISCHER AUSDRUCK 376
ARITHMETISCHES MITTEL 69
ARRAY 53
ASC 24, 97
ASCII 98
ASSEMBLER 111, 112, 129
ASSEMBLER-ANWEISUNG 120
ASSEMBLER-DIREKTIVE 124
AUSDRUCK, ARITHMETISCHER 376
AUSFUEHRUNGSGESCHWINDIGKEIT 198
AUSGABE 10, 43, 44
AUSGABEANWEISUNG 10
AUSPRAEGUNG 53
AUSWERTUNG, STATISTISCHE 67

** B **

B-RAND 366
BACKSPACE 240
BALLSPIEL 171
BASIC 7, 8, 114, 129, 351
BASIC-DIALEKT 8
BEDINGTER SPRUNG 17
BETRIEBSSYSTEM 9, 51
BILDSCHIRMAUSGABE, FORMATIERTE 328
BILDSCHIRMBEREICH 92

BILDSCHIRMBREITE 82
BILDSCHIRMFENSTER 82, 84
BILDSCHIRMGESTALTUNG 322
BILDSCHIRMMASKE 334
BILDSCHIRMRAND 82, 165
BINAERDATEI 127
BINAERSYSTEM 115
BLOAD 128
BREAK 103
BSAVE 128
BUSINESS-GRAPHICS 384
BYTE 53

** C **

CAD 384
CALL 129, 359
CASE 53
CATALOG 26, 62
CHARACTER 359
CHARTMASTER 384
CHR$ 24
CLOSE 56, 374
COBOL 113, 353, 354, 365
COLOR 162
COMPILER 111
COMPUTE 377
CONFIGURATION-SECTION 367
CONT 17, 101, 107
CONTINUE 358
COS 13
CTRL 16, 47
CTRL-B 248
CTRL-C 16, 131
CTRL-D 42, 43, 247
CTRL-F 248
CTRL-I 246
CTRL-M 249
CTRL-N 248
CTRL-O 247
CTRL-P 247
CTRL-Q 249
CTRL-R 249
CTRL-S 17
CTRL-X 249
CTRL-Z 247
CURSOR 142

** D **

DATA 20, 359
DATA-DIVISION 365, 369
DATAN-II 382
DATEI 52, 53, 56
DATEINAME 25, 56
DATEIVERLAENGERUNG 63
DATEN 9
DATENAUSTAUSCH 134
DATENEINGABE 20
DATENMATRIX 53
DATENSATZ 53, 360
DELETE 26, 62
DEZIMALPUNKT 11
DIM 20
DIMENSION 359
DIREKTER SPEICHERZUGRIFF 80
DIREKTER ZUGRIFF 54
DISKETTE 25
DISKETTENLAUFWERK 51
DISKETTENNUTZUNG, INTERAKTIVE 56
DISPLAY 378
DIVIDE 376
DIVISION 365
DO 358
DOKUMENTATION 287
DOS 5, 106
DOS-TOOL-KIT 263, 265
DREIDIMENSIONALE GRAPHIK 186
DRUCKER 41

** E **

EDITIEREN 238
EFFEKT, GRAPHISCHER 161
EINGABE 12
EINGABEANWEISUNG 10
END 14, 360
ENVIRONMENT-DIVISION 365, 367
ESC 238, 239
ESC-1 245
ESC-2 245

ESC-D 239
ESC-E 239
ESC-F 239
ESC-I 239
ESC-J 240
ESC-K 241
ESC-M 241
ESCAPE-MODUS 239
EXEC-DATEI 220
EXP 13

** F **

FARB-INTERFACE 162
FARBDEMONSTRATION 167
FARBE 161, 162, 177
FARBFERNSEHER 162
FARBMONITOR 162
FD 372
FEHLERMELDUNG 145, 386
FELD 53
FIELD 53
FILE 53, 56
FILE-CONTROL 368
FLASH 99, 161
FLUSSDIAGRAMM 35
FOR...TO 19
FORMAT 356, 360
FORMATIERTE BILDSCHIRMAUSGABE 328
FORMATIERUNG 25
FORTRAN 113, 353, 354
FORTRAN-77 354
FORTRAN-IV 355
FUNKTION 13
FUNKTION, LINEARE 180

** G **

GERADE 180
GERAET, PERIPHERES 41
GET 28, 60, 103, 106, 198
GITTER 166

GOSUB 18, 195, 198
GOTO 16, 195, 358, 377
GR 158
GRAPHIK 45, 188
GRAPHIK, DREIDIMENSIONALE 186
GRAPHIK, HOCHAUFLOESENDE 156, 174
GRAPHIK-ANWEISUNG 163, 178
GRAPHIK-AUSGABE 49
GRAPHIKMODUS 158
GRAPHIKPROGRAMM 156
GRAPHISCHER EFFEKT 161

** H **

HAUPTPROGRAMM 18
HCOLOR 177
HEXADEZIMALSYSTEM 117
HGR 176
HIMEM 174
HISTOGRAMM 168
HLIN 164
HOCHAUFLOESENDE GRAPHIK 156, 174
HOME 29, 82, 86
HPLOT 178
HTAB 29, 322

** I **

IDENTIFICATION-DIVISION 365, 366
IF 357, 358, 377
IF...THEN 17, 195, 210
INDIZIERTE VARIABLE 21
INFORMATIONSEINGABE 13
INIT 25, 55, 303
INITIALISIERUNG 25, 55, 303
INPUT 13
INPUT-OUTPUT-SECTION 367
INT 13
INTERAKTIVE DISKETTENNUTZUNG 56
INTERFACE 42
INVERSE 99, 161

** K **

KOMMANDO 9, 15
KOMMENTAR 360
KOORDINATENSYSTEM 179
KORRELATIONSKOEFFIZIENT 70
KREIS 183

** L **

LEFT$ 24, 97
LEN 23
LET 12, 20
LINEARE FUNKTION 180
LINEDOC 263, 288, 295
LIST 15, 81
LOAD 26
LOESCHEN 62, 134
LOG 13
LOMEM 174

** M **

MASCHINENPROGRAMM 80, 112
MASCHINENSPRACHE 111, 130, 140
MASKENGENERATOR 337
MENUE 209
MENUETECHNIK 209
METHODE, STATISTISCHE 69
MID$ 24, 97
MISCHEN 188, 220, 265
MITTEL, ARITHMETISCHES 69
MITTELWERT 69
MON 61
MONITOR 129, 131
MOVE 375
MULTIPLAN 381
MULTIPLY 376

** N **

NEW 15
NEXT 19
NOMON 61
NORMAL 99, 161
NORMALGRAPHIK 156, 158

** O **

OBJECT-PROGRAM 112
ON...GOSUB 209, 211
ON...GOTO 209
OPEN 56, 72, 374

** P **

PARAGRAPH 365
PEEK 81, 82
PERIPHERES GERAET 41
PICTURE 370
PICTURE-KLAUSEL 370
PLOT 163
POKE 81
POSITIONIERUNG 322
PR# 42
PRINT 10, 81, 357
PRINT-USING 328
PROBLEMANALYSE 30
PROCEDURE-DIVISION 365, 374
PRODOS 5
PROFIT-II 384
PROGRAM-ID 366
PROGRAM-LINE-EDITOR 6, 238, 245
PROGRAMM 8
PROGRAMMABLAUF 195
PROGRAMMABLAUFPLAN 35
PROGRAMMANWEISUNG 9
PROGRAMMIERHILFEN . 263
PROGRAMMIERSPRACHE 353
PROGRAMMSCHLEIFE 19
PROGRAMMSTRUKTUR 195
PROGRAMMVERZWEIGUNG 16

** Q **

QUELLENPROGRAMM 112
QUICKFILE 384

** R **

RANDOM-ACCESS-SPEICHERUNG 52, 54
READ 20, 356, 374
RECORD 53
RENAME 26
RENUMBER 280
REPLACE 288
REPTEST 382
RESET 50
RESTORE 22
RETURN 18, 198
RETYPE 240
RIGHT$ 24, 97
RND 13
RUN 15

** S **

SATZ 53
SAVE 26, 62
SCHNITTSTELLE 42
SCHRITTWEITE 19
SCRIPT 383
SEKTION 367
SELECT 369
SEQUENTIELLE SPEICHERUNG 52, 54
SET 53
SIN 13
SOFTWARE 351, 380
SORTIEREN 30, 38, 342
SOURCE-PROGRAM 112
SPALTENINDEX 22
SPEICHERADRESSE 391
SPEICHEREINTEILUNG 393

SPEICHERFELD 10
SPEICHERINHALT 134
SPEICHERUNG, SEQUENTIELLE 52, 54
SPEICHERZUGRIFF 80
SPEICHERZUGRIFF, DIREKTER 80
SPRUNG, BEDINGTER 17
SPRUNGADRESSE 295
SPRUNGANWEISUNG 16
STANDARDABWEICHUNG 69
STARTADRESSE 130
STATEMENT 9
STATISTISCHE AUSWERTUNG 67
STATISTISCHE METHODE 69
STOP 27, 101, 107
STOP-RUN 378
STR$ 24, 97
STRICH 164
STRING 10, 12, 23
STRINGBEARBEITUNGSFUNKTION 23
STRINGFUNKTION 325
STUFENKONZEPT 369, 370
SUBROUTINE 18, 359
SUBTRACT 376
SYMBOL 189

** T **

TAB 12
TAN 13
TASTATUR 100
TAXONOM 383
TEXT 159, 176, 188
TEXTDATEI 62
TEXTMODUS 158
TON 141
TRENNER 11

** U **

UEBERBLICK 287
UMNUMERIEREN 265
UNTERPROGRAMM 18, 140, 197
UNTERPROGRAMMADRESSE 390
UNTERPROGRAMMTECHNIK 197

** V **

VAL 23
VARDOC 263, 288, 290
VARIABLE 53
VARIABLE, INDIZIERTE 21
VARIABLENKONTROLLE 289
VARIABLENNAME 12
VERAENDERUNG 62
VERIFIZIERUNG 135
VERLAENGERUNG 62
VERZWEIGUNG 137
VISICALC 381
VLIN 164
VTAB 29, 160, 322

** W **

WAIT 107, 198
WARTEPROGRAMM 148
WORDSTAR 383
WORKING-STORAGE-SECTION 372
WRITE 357, 375

** X **

X-SORTER 343
X/Y-SORTER 343

** Z **

ZAEHLEN 37
ZEICHNEN 189
ZEILENINDEX 22
ZUGRIFF, DIREKTER 54

ACHTUNG AUTOREN

SIE haben ein gutes Programm oder ein Manuskript zu einem interessanten Buch geschrieben oder würden dies gerne tun

SIE würden das Ergebnis Ihrer Arbeit gerne in größeren Stückzahlen vermarktet sehen

SIE suchen dafür den leistungsfähigen Verlag und Vertriebspartner Ihres Vertrauens

WIR besitzen große Erfahrung in der professionellen Vermarktung von Software und Literatur, nicht nur in Deutschland, sondern auch weltweit

WIR haben in den letzten 12 Monaten mit dem Verkauf von 200.000 Büchern und 50.000 Programmen unsere Leistungsfähigkeit unter Beweis gestellt

WIR suchen weitere Autoren, mit denen wir gemeinsame Erfolge erringen können

SIE und **WIR** sollten zusammenarbeiten.

Bitte, senden Sie eine Beschreibung Ihres Programms oder Ihres Buchprojekts an Dr. Achim Becker c/o DATA BECKER, oder fordern Sie einfach unsere unverbindlichen „Informationen für Autoren" an.

IHR GROSSER PARTNER FÜR KLEINE COMPUTER

DATA BECKER

Merowingerstraße 30 · 4000 Düsseldorf · Telefon (02 11) 31 00 10 · im Hause AUTO BECKER

FANTASTISCH

Was so ein COMMODORE 64 mit DATA BECKER PROGRAMMEN alles kann:

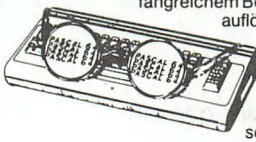
Mit ***DATAMAT*** „frißt" Ihr C-64 Ordner, Karteikästen und Notizbücher. DATAMAT ist eine universelle Dateiverwaltung, die Sie auf vielfältige Weise nutzen können. Frei gestaltbare Eingabemaske mit bis zu 50 Feldern, max. 40 Zeichen pro Feld und bis zu 253 Zeichen pro Datensatz. Bis zu 2000 Datensätze pro Diskette. Sortiermöglichkeit nach mehreren Feldern in beliebiger Kombination. Druck von Auswertungen, Listen und Etiketten. DATAMAT sollte zu jedem 64er gehören.

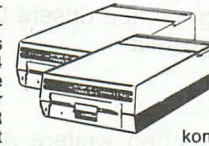
Mit ***TEXTOMAT*** werden Briefe, Rundschreiben und komplette Bücher zum Kinderspiel. TEXTOMAT schafft 80 Zeichen pro Zeile durch horizontales Scrolling, Ausdruck bis 255 Zeichen Breite, Textlänge bis zu 24 000 Zeichen im Speicher, Verketten von Texten, Textbausteinverarbeitung, Formatierung, Blocksatz, Formularsteuerung, Serienbriefe und natürlich deutsche Zeichen nicht nur auf dem Bildschirm, sondern mit vielen Druckern (Epson, GP 100 VC, 1525, 1526, 801) auch auf dem Papier. Mit TEXTOMAT macht Schreiben Spaß.

SYNTHIMAT verwandelt Ihren COMMODORE 64 in einen professionellen, polyphonen, dreistimmigen Synthesizer, der in seinen unglaublich vielen Möglichkeiten großen und teuren Synthesizern kaum nachsteht. Mit SYNTHIMAT wird Ihr 64 für wenig Geld zur Supermaschine.

Entdecken Sie die faszinierende Welt der Computergraphik mit ***SUPERGRAPHIK 64,*** der starken Befehlserweiterung mit den vielseitigen Möglichkeiten. 187 (!) Befehlskombinationen für Sprites, Graphik und Sound. Mit der SUPERGRAPHIK 64 machen Sie mehr aus Ihrem COMMODORE 64. Für Druckerbesitzer gibt es die Möglichkeit, eine Hardcopy des Bildschirms zu erstellen.

PASCAL 64 ist ein leistungsfähiger PASCAL-Compiler, mit umfangreichem Befehlssatz, der auch die hochauflösende Graphik und die Sprites des COMMODORE 64 unterstützt. Ein-/Ausgabe über Diskette und Drucker sowie REAL und INTEGER Arithmetik. PASCAL 64 ist sehr schnell, da echter Maschinencode erzeugt wird!

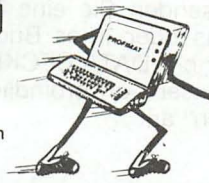




DISKOMAT hilft Ihnen, mehr aus Ihrer Floppy zu machen, mit SUPERTWIN, dem Steuerprogramm, das zwei VC-1541 wie ein Doppellaufwerk verwaltet, mit DISC-BASIC, den Diskettenbefehlen des BASIC 4.0, mit denen Sie eine komplette Diskette oder Auszüge mit einem Befehl kopieren können und mit einem komfortablen DISK-MONITOR.

Mit ***FAKTUMAT*** ist das Schreiben von Rechnungen kein Alptraum mehr. Eine Sofortfakturierung mit integrierter Lagerbuchführung. Individuelle Anpassung von Steuersätzen, Maßeinheiten und Firmendaten. Kunden- und Artikelstamm voll pflegbar. Schneller Zugriff auf Kunden- und Artikeldaten über frei definierbaren, 6-stelligen Schlüssel. Automatische Fortschreibung von Artikel- und Kundendaten, individuell nutzbar. Alles in allem die Arbeits- und Zeitersparnis die Sie sich schon längst gewünscht haben.

Mit Maschinensprache geht vieles schneller. ***PROFIMAT*** enthält den komfortablen Maschinensprache Monitor PROFI-Mon und PROFI-Ass, einen sehr leistungsfähigen Assembler. PROFI-Ass bietet unter anderem formatfreie Eingabe, komplette Assemblerlistings, ladbare Symboltabellen (Labels), redifinierbare Symbole, eine Reihe von Assembleranweisungen, bedingte Assemblierung und Assemblerschleifen.



Darauf haben Sie bestimmt gewartet: Passend zum Superhit COMMODORE 64 gibt's jetzt die neuen DATA BECKER PROGRAMME - Spitzensoftware auf Diskette zu unglaublich niedrigen Preisen. Alle Programme werden mit ausführlichem Handbuch und in einem formschönen Ordner geliefert.

PROFIMAT IN STICHWORTEN

PROFIMAT ist ein leistungsfähiges Programmpaket zur professionellen Maschinenprogrammierung mit dem COMMODORE 64

- PROFI-MAT
- PROFI-MON
- 2 K-Byte Maschinenprogramm
- Register- und Speicherinhalte anzeigen/ändern
- Maschinenprogramme laden, speichern, ausführen, disassemblieren
- Speicherbereiche durchsuchen, vergleichen, füllen
- PROFI-ASS
- 4 K-Byte Maschinenprogramm
- formatfreie Eingabe
- komplette Assemblerlistings
- ladbare Symboltabellen
- bedingte Assemblierung und Assemblerschleifen
- Verkettung von Quellprogrammen
- umfangreiche Pseudo-Opcodes
- vielfältige arithmetische und logische Operationen
- arbeitet mit beliebiger Gerätekonfiguration
- Diskettenprogramm
- mit ausführlichem Handbuch

PASCAL 64 IN STICHWORTEN

PASCAL 64, DAS IDEALE PASCAL FÜR EINSTEIGER ZUM COMMODORE 64

- erzeugt echte Maschinensprache
- ca. 22 K frei für Programm und Symboltabelle
- einfache Programmerstellung durch Bildschirmeditor
- enthält alle wichtigen Methoden zur Programmstrukturierung

Standard PASCAL-Features:
- BEGINN . . . END
- WHILE . . . DO
- REPEAT . . . UNTIL
- CASE
- PROCEDURE
- FUNKTION

Datentypen:
- CHAR
- INTEGER
- REAL
- ARRAY

Sonderfunktionen:
- PEEK
- SYS
- Graphikbefehle
- Behandlung sequentieller Dateien
- einbinden externer Prozeduren und Funktionen von Diskette
- lieferbar auf Diskette
- mit detailliertem Handbuch

SYNTHIMAT IN STICHWORTEN

SYNTHIMAT ist ein leistungsfähiger, polyphoner Synthesizer für den COMMODORE 64

- drei Oszillatoren (VCOs) mit 7 Fußlagen und 8 Wellenformen
- drei Hüllkurvengeneratoren (ADSRs)
- ein Filter (VCF) mit 8 Betriebsarten und Resonanzregulierung
- VCF mit Eingang für externe Signalquelle
- ein Verstärker (VCA)
- Ringmodulation mit allen drei VCOs
- 8 softwaremäßig realisierte Oszillatoren (LFOs)
- kräftiger Klang durch polyphones Spielen
- zwei Manuale (Solo und Begleitung)
- speichern von bis zu 256 Klangregistern
- schneller Registerwechsel
- speichern von 9 Registerdateien Diskette
- „Bandaufnahme" auf Diskette durch direktes Spielen
- keine lästige Noteneingabe
- speichern von bis zu 9 „Bandaufnahmen" je Diskette
- integrierte 24 Stunden-Echtzeituhr
- einstellbares PITCH-BENDING
- farblich gekennzeichnete, übersichtlich angeordnete Module
- umfangreiches Handbuch
- läuft mit einem Diskettenlaufwerk
- Diskettenprogramm

DISKOMAT IN STICHWORTEN

DISKOMAT, ein Programmpaket mit SUPERTWIN, DISK-MONITOR und DISK-BASIC zum COMMODORE 64.

- SUPERTWIN, ein Steuerprogramm, mit dem Sie zwei VC 1541 Diskettenlaufwerke wie ein Doppellaufwerk benutzen. Kopieren von Dateien und zwischen zwei Laufwerken Backup-Möglichkeit. Kompatibel zu Programmen, die für Doppellaufwerke gedacht sind.
- DISK-BASIC bietet Ihnen die komfortablen Diskettenbefehle des BASIC 4.0.

APPEND	BACKUP	CATALOG	COLLECT
CONCAT	COPY	DOPEN	DCLOSE
DLOAD	DSAVE	HEADER	RECORD
SCRATCH	DIRECTORY	RENAME	DS & DS $

- DISK-MONITOR
- Lesen eines Blocks von Diskette
- Anzeige und Ändern eines Blocks am Bildschirm
- Schreiben eines Blocks auf Diskette
- Senden von Diskettenbefehlen
- Anzeigen der Fehlermeldung der Diskette
- Diskettenprogramm mit ausführlichem deutschen Handbuch

SUPERGRAPHIK IN STICHWORTEN

SUPERGRAPHIK 64 ist ein umfassendes Graphikprogramm für den COMMODORE 64

— 2 unabhängige, hochauflösende Graphik-Seiten (320 x 200 Punkte)
— 1 Standard Low-Graphik-Seite (80 x 50 Punkte)
— Normalfarben-Graphik (320 x 200 Punkte)
— Multicolor-Graphik (160 x 200 Punkte)
— verdecktes Zeichnen (z. B. Text sichtbar, Graphikseite 2 wird erstellt)
— 183 Befehle und Befehlskombinationen:

1. Für jeden Befehl wählbare Zwischenmodi: Zeichnen, Löschen, Punktieren, Graphik-Cursor bewegen, Zeichnen mit/ohne Farbsetzung, Punkte zählen
2. Durch einfache Befehle zu steuernde Graphikfiguren: Punkt, Linie, Linienschar, Linie vom Graphik-Cursor, Kreise, Kreisbögen, Ellipse, Ellipsenbögen, selbstdefinierbare Figuren, rotieren und vergrößern dieser Figuren, Rahmen, Feld, Text in Graphik
3. Weitere Graphik-Befehle: Graphikseiten- und Moduswechsel, Graphik löschen, Graphik invertieren, Scrolling von Text und Graphik, Wählen der Rahmen-, Hintergrund-, Zeichen- oder Punktfarbe

— Speichern, Laden von Graphik (auch verdeckt)
— Hardcopies für EPSON, CBM 1526, Seikosha GP 100 VC, Farb(!)drucker Seikosha GP 700 und andere mit DATA BECKER Interface
— 8 Sprites definiert durch einen Befehl!
— alle Sprite-Eigenschaften veränderbar
— Positionieren und Bewegen (!) von 8 Sprites gleichzeitig und unabhängig voneinander, während das übrige Programm weiterläuft (IRQ)
— Sprite-Kollisionsüberprüfung, Joystickunterstützung
— automatische Unterbrechung des BASIC-Programms bei Kollisionen (Interrupt), Sprung in Unterbrechungsroutine, dann Weiterführung des Hauptprogramms
— komfortable Soundprogrammierung mit Verstellung aller möglichen Sound-Parameter (Laufstärke, Klang, Filter, Tonhöhe, Tonlänge) ebenfalls unabhängig vom übrigen Programmablauf
— umfangreiche Anleitung
— Diskettenprogramm

DATAMAT IN STICHWORTEN

DATAMAT ist ein komfortables Dateiverwaltungsprogramm für den COMMODORE 64

— menuegesteuert, dadurch extrem einfach zu bedienen
— für jede Art von Daten zu gebrauchen
— völlig frei gestaltbare Eingabemaske
— 50 Felder pro Datensatz
— 253 Zeichen pro Datensatz
— je nach Umfang der Datensätze bis zu 2000 pro Datei
— Schnittstelle zu TEXTOMAT
— lieferbar als Diskettenprogramm mit ausführlichem deutschen Handbuch
— läuft mit ein oder zwei Floppies

Sie können:

— nach beliebigen Feldern selektieren
— nach allen Feldern gleichzeitig sortieren
— Listen in völlig freiem Format drucken
— Etiketten drucken
— mit jedem COMMODORE Drucker arbeiten
— nahezu jeden Fremddrucker verwenden (mit DATA BECKER Interface)

TEXTOMAT IN STICHWORTEN

TEXTOMAT, ein außergewöhnliches Textverarbeitungsprogramm für den COMMODORE 64

— Diskettenprogramm
— mit umfangreichem deutschen Handbuch
— durchgehend menuegesteuert
— deutscher Zeichensatz auch auf Commodore-Druckern
— Rechenfunktionen
— 24000 Zeichen pro Text
— beliebig lange Texte durch Verknüpfung
— frei programmierbare Steuerzeichen
— horizontales Scrolling für 80 Zeichen pro Zeile
— läuft mit ein oder zwei Floppies
— Formularsteuerung für Randeinstellung usw.
— komplette Bausteinverarbeitung
— Blockoperationen, Suchen und Ersetzen
— Serienbriefschreibung mit DATAMAT
— formatierte Ausgabe auf Bildschirm
— an fast jeden Drucker anpassbar

STRUKTO 64 IN STICHWORTEN

STRUKTO 64 ist eine fantastische neue Programmiersprache für strukturiertes Programmieren mit COMMODORE 64.

— Interpretersprache, die die Vorzüge von BASIC UND PASCAL vereint
— strukturiertes Programmieren
— übersichtliche Programme
— leichte Erlernbarkeit
— einfache Bedienung
— eingebautes Toolkit erleichtert das Eingeben und Verbessern von Programmen
— leichteres Arbeiten mit der Floppy
— Sprite-Editor ermöglicht das Einlesen der Sprite-Formen direkt vom Bildschirm
— Graphikbedienung wird mit gut durchdachten Befehlen unterstützt
— Abspielen von Musik ist unabhängig vom Programmablauf möglich
— ca. 80 neue Befehle
— lieferbar als Diskettenprogramm
— ausführliches, deutsches Handbuch

STRUKTO 64 ist für alle Programmierer geeignet, die den COMMODORE 64 als Allround-Computer einsetzen wollen und auf einfache Weise anspruchsvolle Programme erstellen wollen.

KONTOMAT IN STICHWORTEN

KONTOMAT ein Diskettenprogramm zur Einnahme-/Überschußrechnung

- professionelles Programm, angelehnt an eine echte Finanzbuchhaltung
- maximal 120 definierbare Konten
- 4 Mehrwert- und Vorsteuerkennzeichen
- intervallmäßige Belegeingabe
- 4 verschiedene Buchungsarten (SOLL/HABEN, HABEN/SOLL, SOLL, HABEN)
- Anzeige der SOLL- und HABEN-Summe bei mehrfachen Buchungssätzen
- komfortable Belegeingabe mit Datum, Belegnummer, Buchungstext, Steuerkennzeichen, Buchungsart und Betrag
- Druck des Journals mit der Belegeingabe
- Druck von Kontenblättern
- Druck einer Summen- und Saldenliste mit Monats- und Jahresumsatzsummen
- betriebswirtschaftliche Auswertung mit Druckausgabe
- Verzeichnis der AfA und der Sofortabschreibung
- ein oder zwei Laufwerke
- umfangreiches Handbuch

FAKTUMAT IN STICHWORTEN

FAKTUMAT ist ein Programm zur Fakturierung und Rechnungserstellung für den COMMODORE 64

- voll menuegesteuert
- Artikeldatei & Kundendatei parameterisiert
- läuft auf 1 oder 2 Floppies 1541
- Drucker: VC-1525, VC-1526, EPSON RX-/FX-80 mit DATA BECKER Interface
- Diskettenprogramm
- frei definierbarer Firmenkopf
- 4 wählbare Mehrwertsteuersätze
- Maximal 950 Artikel bei 50 Kunden oder 50 Artikel bei 950 Kunden. Die Summe von Kunden + Artikel kann maximal 1000 betragen
- Druck von Rechnung und Lieferschein möglich
- Lagerbuchführung ist integriert. Alle Artikel die über Rechnung/Lieferschein rausgegangen sind, werden in der Lagerdatei abgebucht.
- Rechnungsbeträge werden in der Kundendatei festgehalten
- deutsches, detailliertes Handbuch
- Sie arbeiten mit einer Diskette für Ihre Daten

PAINT-PIC IN STICHWORTEN

PAINT-PIC ist ein faszinierendes Malprogramm für den COMMODORE 64

- Programmsteuerung: Tastatur
- Steuerung des Stifts: Cursortasten und eckige Klammer (diag.) (Joystick kann benutzt werden)
- Routinen: Linien, Rechtecke, Dreiecke, Parallelogramme, Kreise, Kreisbögen, Ellipsen, Bestimmung v. Mittelpunkt, u. perspektivischer Linie, Kopieren u. drehen v. Teilbildern, Verdoppeln, halbieren u. spiegeln v. Teilbildern
- Modi: Malstiftmodus (schmale Linie) Pinselmodus (8 verschiedene Breiten) (Art der Linie selbst definierbar)
- Textmodus (Kompl. Zeichensatz Commodore) (Hoch-Tiefschrift)
- Speichern: Teilbilder (Blöcke) oder ganze Bilder
- Menue: 1 Hauptmenue mit 8 Untermenues
- mit ausführlichem deutschen Handbuch
- Diskettenprogramm
- Bilder kann man auf Diskette oder Cassette abspeichern

MASTER 64 IN STICHWORTEN

MASTER: Professionelles Programmentwicklungssystem für COMMODORE 64

- 70 zusätzliche Befehle
- Bildschirmmaskengenerator
- Definieren von Bildschirmzonen
- Eingabe aus Zonen
- Formatierte Ausgabe
- Abspeicherung von Bildschirminhalten
- Arbeiten mit mehreren Bildschirmmasken
- ISAM Dateiverwaltung
- Datensätze bis zu 254 Zeichen
- Schlüssellänge bis zu 30 Zeichen
- Dateigröße nur von Diskettenkapazität abhängig
- Zugriff über Schlüssel und Auswahlmasken
- Druckmaskengenerator
- Erstellung beliebiger Formulare und Ausgabemasken
- BASIC-Erweiterungen
- Toolkitfunktionen
- Mehrfachgenaue Arithmetik Rechnen mit 22 Stellen Genauigkeit

Alles, was Sie schon immer über Ihren COMMODORE wissen wollten!

DAS
MASCHINENSPRACHE-
BUCH ZUM
COMMODORE 64
1984, ca. 200 Seiten,
DM 39,-

Eine leicht verständliche Einführung in die Programmierung in Maschinensprache für alle, denen die Programmierung des Commodore 64 in BASIC nicht mehr ausreicht. Sie lernen Aufbau, Arbeitsweise und Register des 6510-Mikroprozessors kennen und werden über Bit, Bytes, Daten und Adressen aufgeklärt. Anhand von Beispielen lernen Sie alle Befehle des 6510 kennen und anwenden. Dabei werden die Analogien zu BASIC Ihnen beim Verständnis helfen. Ein weiteres Kapitel beschäftigt sich mit der Eingabe von Maschinenprogrammen. Dort erfahren Sie auch alles über Monitor-Programme sowie über Assembler. Zum einfachen und komfortablen Erstellen Ihrer eigenen Maschinensprache enthält das Buch einen kompletten Assembler, damit Sie gleich von Anfang an komfortabel und effektiv programmieren können. Weiterhin finden Sie dort einen Disassembler, mit dem Sie sich Ihre Maschinenprogramme oder die Routinen des BASIC-Interpreters und des Betriebssystems ansehen können. Ein besonderer Clou ist ein in BASIC geschriebener Einzelschrittsimulator, mit dem Sie Ihre Programme schrittweise ausführen können. Dabei werden Sie nach jedem Schritt über Registerinhalte und Flags informiert und können den logischen Ablauf Ihres Programms verfolgen. Eine unschätzbare Hilfe besonders für den Anfänger. Als Beispielprogramm finden Sie ausführlich beschriebene Routinen zur Grafikprogrammierung und für BASIC-Erweiterungen. Natürlich sind alle Beispiele und Programme auf Ihren Commodore 64 zugeschnitten.

DAS TRAININGSBUCH
ZUM SIMON's BASIC,
1984, ca. 300 Seiten,
DM 49,-

SIMON's BASIC ist ein Hit — wenn man es richtig nutzen kann. Deshalb gibt es jetzt zu dieser vielseitigen Befehlserweiterung unser umfangreiches Trainingsbuch. Auf über 300 Seiten erklärt es Ihnen detailliert den Umgang mit den über 100 Befehlen des SIMON's BASIC. Alle Befehle werden ausführlich dargestellt, auch die, die nicht im Handbuch stehen! Natürlich zeigen wir auch die Masken des SIMON's BASIC und geben wichtige Hinweise, wie man diese umgeht. Zum Buch gehören zahlreiche Beispielprogramme und viele interessante Programmiertricks. Alle Befehle sind entsprechend ihren Anwendungen in entsprechende Kapitel geordnet: Einführung in das CBM BASIC 2.0 — Programmierhilfen — Fehlerbehandlung — Programmschutz — Programmstruktur — Variablen — Zahlenbehandlung — Eingabekontrolle — Ein/Ausgabe Peripheriebefehle — Graphik — Zeichensatzerstellung — Sprites — Musik — Steuernde Peripherie — SIMON's BASIC und die Verträglichkeit mit anderen Erweiterungen und Programmen. Dazu ein umfangreicher Anhang. Nach jedem Kapitel finden Sie Testaufgaben zum optimalen Selbststudium und zur Lernerfolgskontrolle. DAS TRAININGSBUCH ZUM SIMON's BASIC sollte jeder Anwender dieser universellen Befehlserweiterung unbedingt haben.

DATA BECKER BÜCHER

DAS GROSSE
FLOPPY-BUCH,
1983, ca. 320 Seiten,
DM 49,-

Darauf haben Sie gewartet: Endlich ein Buch, das Ihnen ausführlich und verständlich die Arbeit mit der Floppy VC-1541 erklärt. DAS GROSSE FLOPPY BUCH ist für Anfänger, Fortgeschrittene und Profis gleichermaßen interessant. Sein Inhalt reicht von der Programmspeicherung bis zum DOS-Zugriff, von der sequentiellen Datenspeicherung bis zum Direktzugriff, von der technischen Beschreibung bis zum ausführlich dokumentierten DOS-Listing, von den Systembefehlen bis zur detaillierten Beschreibung der Programme der Test/Demodiskette. Exakt beschriebene Beispiel- und Hilfsprogramme ergänzen dieses neue Superbuch. Aus dem Inhalt: Der erste Kontakt — Das Speichern von Programmen — Die Floppy-Systembefehle — Sequentielle Datenspeicherung — Relative Datenspeicherung — Die Fehlermeldungen und Ihre Ursachen — Der Direktzugriff — Der Zugriff auf das DOS — Technik der Floppy und der Diskette — DOS-Listing der VC-1541 — Dienstprogramme — BASIC-Erweiterungen und Programme zur komfortablen Benutzung der VC-1541 - Overlaytechnik — Diskmonitor — IEC-Bus und serieller Bus — Gemeinsamkeiten mit den großen CBM-Floppies und Unterschiede gegenüber der VC-1541. Mit dem GROSSEN FLOPPY-BUCH meistern Sie auch Ihre Floppy.

64 FÜR PROFIS
1983, ca. 280 Seiten,
DM 49,-

Wer besser und leichter in BASIC programmieren möchte, der braucht dieses Buch. 64 FÜR PROFIS zeigt, wie man erfolgreich Anwendungsprobleme in BASIC löst und verrrät die Erfolgsgeheimnisse der Programmierprofis. Vom Programmentwurf über Menüsteuerung, Maskenaufbau, Parameterisierung, Datenzugriff und Druckausgabe bis hin zur guten Dokumentation wird anschaulich mit Beispielen dargestellt, wie Profiprogrammierung vor sich geht. Besonders stolz sind wir auf die völlig neuartige Datenzugriffsmethode QUISAM, die in diesem Buch zum ersten Mal vorgestellt wird. QUISAM erlaubt eine beliebige Datenzusatzlänge, die dynamisch mit der Eingabe der Daten wächst. Eine lauffertige Literaturstellenverwaltung veranschaulicht die Arbeitsweise von QUISAM. Neben diesem Programm finden Sie auch noch weitere Programme zur Lager- und Adressenverwaltung, Textverarbeitung und einen Reportgenerator. Alle diese Programme sind mit Variablenliste versehen und ausführlich beschrieben. Damit sind diese für Ihre Erweiterungen offen und können von Ihnen an Ihre speziellen Bedürfnisse angepaßt werden. Mit 64 FÜR PROFIS steigen Sie in die Welt der Programmierprofis ein.

MIT DATA BECKER BÜCHERN MACHEN

SIE MEHR AUS IHREM COMMODORE

DATA BECKER BÜCHER

64 TIPS & TRICKS,
2. Auflage,
1983, ca. 290 Seiten,
DM 49,-

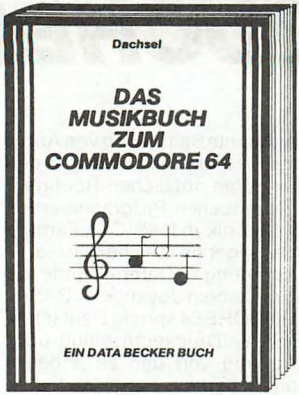
64 Tips & Tricks ist eine hochinteressante Sammlung von Anregungen zur fortgeschrittenen Programmierung des COMMODORE 64, Poke's und anderen nützlichen Routinen, interessanten Programmen sowie aktuellen Programmiertips und -tricks. Aus dem Inhalt: 3D-Graphik in BASIC — Farbige Balkengraphik — Definition eines eigenen Zeichensatzes — Die Tastaturbelegung und ihre Änderung — Dateneingabe mit Komfort — Simulation der Maus mit einem Joystick — BASIC für Fortgeschrittene — Ihr COMMODORE 64 spricht Deutsch — CP/M auf dem COMMODORE 64 — Druckeranschluß über den USER-Port — Datenübertragung von und zu anderen Rechnern — Der Expansionsport — Synthesizer in Stereo — Retten einer nicht ordnungsgemäß geschlossenen Datei — Erzeugen einer BASIC-Zeile in BASIC — Der Kassettenpuffer als Datenspeicher — Sortieren von Stringfeldern — Multitasking auf dem COMMODORE 64 — POKE's und die Zeropage — GOTO, GOSUB und RESTORE mit berechneten Zeilennummern INSTR und STRING-Funktion — Repeatfunktion für alle Tasten — und vieles andere mehr. Alle Maschinenprogramme mit BASIC-Ladeprogrammen. 64 TIPS & TRICKS ist eine echte Fundgrube für jeden COMMODORE 64 Anwender.

64 INTERN,
3. Auflage,
1983, ca. 320 Seiten,
DM 69,-

64 INTERN ist unser großes Buch zum COMMODORE 64. Ideal für alle, die sich näher mit Programmierung, Technik und Betriebssystem von COMMODORE's Supermaschine auseinandersetzen wollen. Detailliert werden Architektur und technische Möglichkeiten des C-64 beschrieben, vom Prozessor 6510 über Video-Controller 6569, Sound Controller 6581, Ein-/Ausgabesteuerung mit CIA 6526 bis hin zum seriellen IEC-Bus, USER-Port und EXPANSION-PORT. Paralell dazu wird die Programmierung der hochauflösenden Graphik und der Sprites erklärt, die Sonderprogrammierung und die Programmierung des C-64 in Maschinensprache. Mit einen ausführlich dokumentierten ROM-Listing wird das Betriebssystem zerlegt. Wertvolle Hinweise zum Umsetzen von Programmen bringt ein Vergleich zwischen VC-20, C-64 und CBM 8000. Zahlreiche lauffertige Beispielprogramme und Routinen, z. B. Graphik-Aid, HARDCOPY, RENEN und PRINT USING runden das Buch ab. Der Clou von 64 INTERN sind neben den zahlreichen Schaltbildern und Blockdiagrammen 2 Original COMMODORE-Schaltpläne zum Ausklappen, die zusätzlich sehr ausführlich beschrieben und dokumentiert sind. Dieses Buch sollte jeder COMMODORE 64- Anwender und Interessent haben.

MIT DATA BECKER BÜCHERN MACHEN

SIE MEHR AUS IHREM COMMODORE



Der COMMODORE 64 ist ein Musikgenie. Mit diesem Buch lernen Sie alles über seine musikalischen Fähigkeiten und wie Sie diese programmtechnisch nutzen. Der Inhalt des Musikbuches reicht von einer Einführung in die Computermusik über die Erklärung der Hardware-Grundlagen und die Programmierung in BASIC bis hin zur fortgeschrittenen Musikprogrammierung in Maschinensprache. Zahlreiche Beispielprogramme, komplette Songs und nützliche Routinen ergänzen den leicht verständlichen Text. Geschrieben wurde das Buch von Thomas Dachsel, dem Autor der weltbekannten Musikprogramme SYNTHIMAT und SYNTHESOUND. Erschließen Sie sich die Welt des Sounds und der Computermusik mit dem Musikbuch zum COMMODORE 64.

DAS MUSIKBUCH ZUM
COMMODORE 64
1984, ca. 200 Seiten, DM 39,—

So etwas haben Sie gesucht: Umfassendes Nachschlagewerk zum COMMODORE 64 und seiner Programmierung. Allgemeines Computerlexikon mit Fachwissen von A-Z und Fachwörterbuch mit Übersetzungen wichtiger englischer Fachbegriffe - das DATA BECKER LEXIKON ZUM COMMODORE 64 stellt praktisch drei Bücher in einem dar. Es enthält eine unglaubliche Vielfalt an Informationen und dient so zugleich als kompetentes Nachschlagewerk und als unentbehrliches Arbeitsmittel. Viele Abbildungen und Beispiele ergänzen den Text. Ein Muß für jeden COMMODORE 64 Anwender.

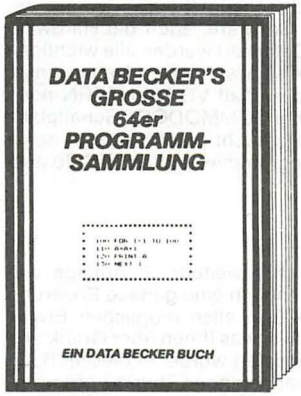
DAS DATA BECKER
LEXIKON ZUM
COMMODORE 64
1984, ca. 350 Seiten, DM 49,—

Graphik ist eine der Hauptstärken des COMMODORE 64. Mit diesem leicht verständlich geschriebenen, aber sehr umfassenden neuen Buch lernen Sie, wie Sie die graphischen Fähigkeiten für Ihre eigenen Programme optimal nutzen. Der Inhalt reicht von den Grundlagen der Graphikprogrammierung über das Erzeugen einfacher Figuren, die Arbeit mit Sprites, Zeichensatzprogrammierung, Hardcopy und IRQ-Handhabung bis hin zur Funktionendarstellung, Laufschrift, Statistik, 3-D, CAD, den Geheimnissen der Actionsspiele und Lightpenanwendungen. Zahlreiche Beispielprogramme, nützliche Routinen und komplette Anwendungen ergänzen diese Buch, das die faszinierende Welt der Computergraphik jedermann zugänglich macht. Geschrieben wurde das Graphikbuch von dem bekannten SUPERGRAPHIK-Autor Axel Plenge.

DAS GRAPHIKBUCH
ZUM COMMODORE 64 1984, ca. 250 Seiten, DM 39,—

Das sollte Ihr erstes Buch zum COMMODORE 64 sein: 64 FÜR EINSTEIGER ist eine sehr leicht verständliche Einführung in Handhabung, Einsatz, Ausbaumöglichkeiten und Programmierung des COMMODORE 64, die keinerlei Vorkenntnisse voraussetzt. Sie reicht vom Anschluß des Geräts über die Erklärung der einzelnen Tasten und Funktionen sowie die Peripheriegeräte und ihre Bedienung bis zum ersten Befehl. Schritt für Schritt führt das Buch Sie in die Programmiersprache BASIC ein, wobei Sie nach und nach eine komplette Adressenverwaltung erstellen, die Sie anschließend nutzen können. Zahlreiche Abbildungen und Bildschirmfotos ergänzen den Text. Viele Anwendungsbeispiele geben nützliche Anregungen zum sinnvollen Einsatz des COMMODORE 64. Das Buch ist sowohl als Einführung als auch als Orientierung vor dem 64er Kauf gut geeignet.

64 FÜR EINSTEIGER
1984, ca. 200 Seiten, DM 29,—



Diese neue, umfangreiche Programmsammlung hat es in sich. Über 50 Spitzenprogramme für den COMMODORE 64 aus den unterschiedlichsten Bereichen, von attraktiven Superspielen („Senso", „Pengo", „Seeschlacht", „Poison Square", „Memory") über Graphik- und Soundprogramme („Fourier 64", „Akustograph", „Funktionsplotter") sowie Utilities („SORT", „Renumber", „Disk Init", „Menue") bis hin zu kompletten Anwendungsprogrammen wie „Videothek", „File Manager" und einer komfortablen Haushaltsbuchführung, in der fast professionell gebucht wird. Der Hit sind zu jedem Programm aktuelle Programmiertips und Tricks der einzelnen Autoren zum Selbermachen. Also — nicht nur abtippen, sondern auch dabei lernen und wichtige Anregungen für die eigene Programmierung sammeln.

DATA BECKER's GROSSE 64er PROGRAMMSAMMLUNG
1984, ca. 250 Seiten, DM 49,—

Achtung Hobbyelektroniker: Dieses neue Buch enthält nicht nur alles über die Ausbaumöglichkeiten des COMMODORE 64 und wie man ihn über seine Schnittstellen mit dem „Rest der Welt" verbindet, sondern auch umfassende Informationen über die vielfältigen Einsatzmöglichkeiten des COMMODORE 64 von der Lichtorgel über Motorsteuerung, Spannungs- und Temperaturmessung bis zur programmierbaren Stromversorgung, und wie man diese verwirklicht. Umfassende Darstellung der Grundlagen. Als Clou enthält das Buch zehn komplette Schaltungen zum Selberbauen, vom Eprommer über Eprom-Karte, Logic Analyzer, Frequenzzähler, Hardware-Tracer, Pulsmeßgerät, Klatschschalter und Digital-Voltmeter bis zur preiswerten Spracheingabe-Sprachausgabe. Jeweils komplett mit Schaltplan, Layout und Softwarelisting. Auch für Profis interessant.

DER COMMODORE 64 UND
DER REST DER WELT 1984, ca. 220 Seiten, DM 49,—

DATA BECKER BÜCHER

VC-20 INTERN,
2. Auflage,
1983, ca. 230 Seiten,
DM 49,-

Die bereits 3. Auflage von VC-20 INTERN ist wieder erheblich erweitert worden. Das Buch beschäftigt sich ausführlich mit der Technik und dem Betriebssystem des VC-20. Dazu gehört natürlich zuerst einmal ein komplettes und ausführlich dokumentiertes ROM-Listing. Dazu gehört auch die Belegung der ZEROPAGE, dem wichtigsten Speicherbereich für den 6502-Prozessor, eine übersichtliche Auflistung der Adressen aller Betriebssystemroutinen, ihrer Bedeutung und ihrer Übergabeparameter. Dies ermöglicht dem Programmierer endlich, den VC-20 von Maschinensprache aus sinnvoll einzusetzen. Denn warum Routinen, die bereits vorhanden sind, noch einmal schreiben? Für Einsteiger befindet sich eine Einführung in die Programmierung in Maschinensprache am Anfang des Buches. Diese behandelt die Themen Maschinensprachemonitor, Disassembler und Assembler. Außerdem wird die Verbindung von BASIC- und Maschinenspracheprogrammen besprochen. Doch nicht nur die Software, auch die Hardware wird ausführlich beschrieben. Detailliert werden alle wichtigen IC's im VC-20 in Ihrer Arbeitsweise beschrieben. Neben einem übersichtlichen Blockschaltbild enthält VC-20 INTERN noch als besonderen Clou drei Original COMMODORE Schaltpläne zum Ausklappen. Dieses Buch braucht jeder, der sich näher mit Technik und Maschinenprogrammierung des VC-20 auseinandersetzen möchte.

VC-20 TIPS & TRICKS,
2. Auflage,
1983, ca. 230 Seiten,
DM 49,-

Die überarbeitete und erheblich erweiterte 2. Auflage von VC-20 TIPS & TRICKS enthält endlich eine genaue Erklärung des Speicheraufbaus des VC-20 in allen möglichen Erweiterungsstufen. Vergessen Sie alles, was Ihnen über Grafik und Sound auf dem VC-20 bisher geliefert wurde. In diesem Buch finden Sie von den Grundlagen angefangen über die programmierbaren Zeichen, über eine Grafikerweiterung in Maschinensprache, über luxuriöse Programme wie z. B. Grafikeditor, Funktionenplotter oder 3D-Grafiken bis hin zur Supererweiterung zum VC-20 alles Wissenswerte und zum Arbeiten Notwendige für eine gute Grafik. Ebenso ausführlich wird die Tonerzeugung behandelt. Neben der Erläuterung der Grundlagen und einem Soundeditor sind natürlich auch fertige Programme, z. B. der VC-20 als Synthesizer oder Schlagzeug abgedruckt. Auf über 100 Seiten finden Sie viele nützliche Tricks, POKE-Befehle, BASIC-Erweiterungen fix und fertig zum Eintippen. Dazu gehören z. B. Unnew, automatische Zeilennumerierung, die Programmierung des User-Port, ein Diskmenue, Hardcopy und so weiter. Bringen Sie Ihrem VC-20 völlig neue Befehle bei! Abgerundet wird das Buch durch Anwenderprogramme zum Eintippen, wie Dateiverwaltung, Textverarbeitung und Spiele. VC-20 TIPS & TRICKS braucht jeder VC-20 Besitzer, der mehr über seinen Computer wissen möchte.

DAS STEHT DRIN:

APPLE II TIPS & TRICKS bietet eine ungeheure Fülle an Informationen, die dem schon etwas erfahrenen APPLE II-Besitzer viele zusätzliche Möglichkeiten eröffnet.

Aus dem Inhalt:

- Peripherieanwendung
- Interaktive Diskettennutzung
- Mischen von Texten und Graphik
- Menuetechnik
- Direkter Speicherzugriff
- Unterprogramme in Maschinensprache
- Hochauflösende Grafik
- Mischen von Text und Grafik
- Mischen von Programmen
- Sortieren
- Speichereinteilung

UND GESCHRIEBEN HABEN DIESES BUCH:

Dr. Renate Prust ist wissenschaftliche Mitarbeiterin an der Universität Bochum. Nebenberufliche Tätigkeit im Bereich der EDV.

Dr. Werner Voß ist Professor für Statistik an der Universität Bochum. Nebenberufliche Tätigkeit im Bereich der Datenverarbeitung und Programmierung.

ISBN 3-89011-025-8